PhD THESIS

INTEGRATION OF VIRTUAL PROGRAMMING LAB IN A PROCESS OF TEACHING PROGRAMMING EDUSCRUM BASED

José Marílio Oliveira Cardoso

ESCUELA DE DOCTORADO INTERNACIONAL
PROGRAMA DE DOCTORADO EN INVESTIGACIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

SANTIAGO DE COMPOSTELA
2020

DECLARACIÓN DO AUTOR/A DA TESE

Integration of Virtual Programming Lab in a process of teaching programming EduScrum based

D./Dna. .José Marílio Oliveira Cardoso

Presento a miña tese, seguindo o procedemento axeitado ao Regulamento, e declaro que:

1) A tese abarca os resultados da elaboración do meu traballo.
2) De selo caso, na tese faise referencia ás colaboracións que tivo este traballo.
3) A tese é a versión definitiva presentada para a súa defensa e coincide coa versión enviada en formato electrónico.
4) Confirmo que a tese non incorre en ningún tipo de plaxio doutros autores nin de traballos presentados por min para a obtención doutros títulos.

En Valongo, 14 de Novembro de 2020

Asdo…………

AUTORIZACIÓN DEL DIRECTOR/TUTOR DE LA TESIS

D./Dña. António Abel Vieira de Castro

En condición de: **Director/a**

Título de la tesis: Integration of Virtual Programming Lab in a process of teaching programming EduScrum based

INFORMA:

Que la presente tesis, se corresponde con el trabajo realizado por D/Dña José Marílio Oliveira Cardoso, bajo mi dirección/tutorización, y autorizo su presentación, considerando que reúne los requisitos exigidos en el Reglamento de Estudios de Doctorado de la USC, y que como director/tutor de esta no incurre en las causas de abstención establecidas en la Ley 40/2015.

En **Porto**, **31 de octubre de 2020**

Firma electrónica

AUTORIZACIÓN DEL DIRECTOR/TUTOR DE LA TESIS

D./Dña. Álvaro Manuel Reis da Rocha

En condición de: **Director/a**

Título de la tesis: Integration of Virtual Programming Lab in a process of teaching programming EduScrum based

INFORMA:

Que la presente tesis, se corresponde con el trabajo realizado por D/Dña José Marílio Oliveira Cardoso, bajo mi dirección/tutorización, y autorizo su presentación, considerando que reúne los requisitos exigidos en el Reglamento de Estudios de Doctorado de la USC, y que como director/tutor de esta no incurre en las causas de abstención establecidas en la Ley 40/2015.

En **Porto**, **31 de octubre de 2020**

Firma electrónica

AUTORIZACIÓN DEL DIRECTOR/TUTOR DE LA TESIS

D./Dña. Antonio Jesús García Loureiro

En condición de: **Tutor/a**

Título de la tesis: Integration of Virtual Programming Lab in a process of teaching programming EduScrum based

INFORMA:

Que la presente tesis, se corresponde con el trabajo realizado por D/Dña José Marílio Oliveira Cardoso, bajo mi dirección/tutorización, y autorizo su presentación, considerando que reúne l os requisitos exigidos en el Reglamento de Estudios de Doctorado de la USC, y que como director/tutor de esta no incurre en las causas de abstención establecidas en la Ley 40/2015.

En **Santiago de Compostela**, **31 de octubre de 2020**

Firma electrónica

To my parents

To my children Pedro and Miguel

To Virgínia

ACKNOWLEDGMENTS

Some say that writing the statement of appreciation is one of the hardest and most difficult tasks in preparing a thesis. They are, I am sure, quite right. Acknowledgments are manifestly incomplete and insufficient, if for no other reason, because of the injustice to some of those who had an influence on this outcome and because others, though not forgotten, are not mentioned explicitly.

For legal reasons, this thesis is presented under the name of a single author. Nevertheless, this work is the culmination of a process that has many antecedents and has been influenced by many people who, explicitly or implicitly, consciously or less consciously, voluntarily or not, have contributed to the personal and professional training of the author and have helped him get here. There are, however, some people who deserve special mention and whose action was crucial.

Of these, I would like to point out, in a very significant way, my advisors. I am deeply grateful to Professor António Vieira de Castro who challenged me, who believed in me and pushed me towards this outcome. For his support, dedication and patience, for valuable suggestions, criticisms, teachings and revisions throughout a journey filled with highs and lows. It is to him that a lot of this work is owed. Above all, I thank him for the pleasure of his company and good spirits and for the honour of his friendship, which I hold in high esteem.

Also, to Professor Álvaro Rocha I pay my deepest gratitude for his fundamental action in the beginning of this process, as well as for his very valuable advice, suggestions, criticism and questioning throughout the development of this work. I am also grateful for the revisions made and for his availability and commitment to supporting my work, to which he contributed decisively with his actions, experience and knowledge.

I also thank Professor Antonio García Loureiro, my tutor, for his follow-up and help throughout the process.

To the (many) colleagues of the Instituto Superior de Engenharia do Porto (ISEP) who encouraged me and in the most varied ways (some without knowing it) contributed to making this work possible, who worried about me in this process and asked me about its progress, motivating me to progress. I am sure you share with me the joy of this outcome. Especially: Alberto Sampaio, Álvaro Teixeira, Ana Almeida, Ana Barata, Ana Cristina Meira, Ana Madureira, Bertil Marques, Constantino Martins, Elsa Gomes, Emanuel Silva, Isabel Sampaio, Jorge Duarte, Maria João Viamonte, Nélson Freire, Nídia Sá Caetano, Nuno Bettencourt, Paulo Ferreira, Piedade Carvalho, Ricardo Almeida, Rosa Reis, Sandra Luna and Sérgio Moreira.

I would like to thank Alexandre Gouveia, Paula Tavares and Rosa Barroso in a very special way for their availability and priceless help in carrying out the experiments, as well as for their camaraderie.

To Dulce Mota for her valuable advice and unconditional support, to Nuno Morgado for his readings and suggestions, to Ana Catarina Fernandes for her thorough review and, especially to Bertil Marques for all support to close this latest version.

To Rui Marques I pay my deepest appreciation and gratitude for his precious availability and help, his encouragement and companionship.

I thank Jorge Mendonça for his important clarifications and help in the processing of the data, in particular for all the time he has given me in this task.

I thank Luiz Faria for his availability, institutional support and technical help, and Conceição Neves and Ângelo Martins for their confidence in me, believing in the success of the experiment and for having granted authorization for it to be carried out.

I am grateful to ISEP for granting permission to install the VPL plugin in the institutional Moodle and to Horacio Macedo for his action in installing and supporting the installation. I must also mention the teaching leave I was granted for a semester, which was essential to the completion of this work.

To those people who, not having directly contributed to this work, were, however, very important throughout my academic and professional life, and who also have a share of responsibility in my work. I would like to highlight my friend Luís Castanheira and Professor Zita Vale who was responsible for starting my career in higher education.

I also leave a special thanks to the students, the reason for the existence of the school, and the final recipients of this work, in particular those who actively participated in the tasks developed, who with their activity, opinions and suggestions have unequivocally enriched the results of this study.

To my parents who, without taking away (all) the stones from my path, have provided me with the conditions and have given me the tools to get here.

To my children and to my wife, not only for the encouragement, but mainly for the time spent and that was owed to them.

To all those who, in various ways, have contributed to making this work possible, my sincere thanks.

ABSTRACT

Programming is, nowadays, a necessary and important skill for any professional in both technical and technological areas. Teaching programming is essential for computer applications development and for technological evolution. However, learning (and teaching) to program is a difficult process with many singularities.

The programming teaching model is usually based on a set of tasks that generally cause some difficulty to students, particularly to novice ones. The efficient way to learn to program is by programming and hard training and thus feedback is a crucial factor in the success and flow of the process. In this setting, automatic code evaluation tools can be important to help teaching (and learning) how to program. One of those tools is Virtual Programming Lab (VPL), a Modular Object-Oriented Dynamic Learning Environment plugin (Moodle).

This doctoral thesis aims to analyze the potential use of VPL in the teaching process of programming in higher education. It also aims to evaluate whether if, with VPL, is possible to make students' learning more effective and autonomous and, at the same time reducing the teaching workload in the evaluation process.

Real-world experiments were carried out with the introduction of VPL in a teaching-learning process supported by Java programming. The teaching method uses the eduScrum methodology, usually supported by Moodle, and it is used in a course of programming initiation of the Degree in Informatics Engineering (LEI) of the Informatics Engineering Department of School of Engineering (ISEP), Polytechnic Institute of Porto (P.PORTO). In APROG the eduScrum methodology is used, usually supported by Moodle, and the Java programming language is used for coding. In the experiments performed the VPL was used in the teaching-learning process allowing students to automatically validate their code.

With this model, we intended to speed up the teaching-learning process by reducing the assessment time and enhancing the assessment of tasks performed by students.

In this study the data was collected directly from observation during the performance of activities during classes, from automatic Moodle registrations and, mainly, from responses to an anonymous online survey.

The results show that the use of VPL in programming curricular units can be an important ally for students and teachers, allowing for more autonomous learning and at the student's pace, and assisting teachers in the process to verify code and giving feedback, giving them more time to support students.

The results of the experiments carried out, supported by the subjects' responses to surveys, point towards the validity of this model.

Keywords

Teaching programming, Virtual Programming Lab (VPL), eduScrum, real time validation, automatic assessment, innovation.

RESUMO AMPLIADO

MARCO

A sociedade actual depende moito da tecnoloxía, como demostra o gran número de equipos e dispositivos presentes nos contextos máis diversos. A electricidade, as telecomunicacións e internet son unha realidade tan constante e adquirida que hoxe son bens esenciais na nosa vida diaria e indispensables para o funcionamento da sociedade.

A tecnoloxía en xeral e as tecnoloxías da información en particular están presentes en calquera área de actividade, como a saúde, o transporte, a educación, a banca ou calquera outra actividade económica. Esta realidade ten un impacto directo na forma en que nos comunicamos, viaxamos, estudamos, traballamos e interactuamos, e a informática xoga un papel crucial no desenvolvemento da sociedade moderna (Fiolhais, 2005).

Os sistemas informáticos son cada vez máis complexos e requiren compoñentes físicos, normalmente denominados *hardware,* como ordenadores, servidores, sistemas de almacenamento ou infraestrutura de rede. Non obstante, para que os sistemas funcionen correctamente, necesitan programas que lles permitan controlar e comunicarse co *hardware* para realizar as operacións previstas, é dicir, o *software*. O *software* consiste nun conxunto de instrucións escritas nunha linguaxe de programación, é dicir, unha linguaxe formal cun conxunto de regras sintácticas e semánticas e as súas propias especificacións.

Ao longo da historia, desenvolvéronse numerosas linguaxes de programación, para fins específicos ou xerais, compilados, interpretados ou híbridos, para sistemas propietarios ou abertos, hai linguaxes de programación para diversos propósitos e contextos (Parker & Davey, 2012) (Kumar & Dahiya, 2017).

A programación encárgase de "traducir" a formulación da lóxica de resolución de problemas a unha linguaxe de programación. Antes de escribir as instrucións, é necesario producir o algoritmo, que pode definirse como unha secuencia de pasos que recibe valores de entrada e leva a un valor ou conxunto de valores como saída (Cormen, Leiserson, Rivest, & Stein, 2009).

Aínda que a produción de *software* non se trata só de programar, esta é unha tarefa esencial para o éxito do proceso e é necesario formar novos programadores e enxeñeiros de *software* para garantir o funcionamento e o desenvolvemento de sistemas informáticos. Polo tanto, a programación docente é unha actividade moi importante para a evolución tecnolóxica e a sustentabilidade da sociedade actual.

Ensinar a codificar é unha tarefa desafiante para calquera profesor, e é aínda máis complexo á hora de ensinar aos alumnos principiantes. Tamén para a maioría dos estudantes, a introdución á programación é unha tarefa difícil (Moström, 2011), con taxas de fracaso xeralmente altas nas disciplinas de programación introdutoria de todo o mundo, en todos os sistemas e a calquera nivel educativo (Gomes A. , 2010).

PROBLEMA DE INVESTIGACIÓN E OBXECTIVOS

No Grao en Enxeñaría en Informática (LEI) do Departamento de Enxeñaría en Informática (DEI) do Instituto Superior de Enxeñaría do Instituto Politécnico de Porto (ISEP), o curso de programación e algoritmo (APROG) realízase no primeiro semestre do primeiro ano. Este curso ten como obxectivo proporcionar aos estudantes iniciantes a competencia esencial para un futuro enxeñeiro en informática: a programación.

En APROG hai clases teóricas (T), teórico-prácticas (TP) e de laboratorio (PL). As clases T son para a presentación de conceptos, as clases TP para demostración e exemplificación da aplicación dos conceptos presentados e as clases PL para adestramento de resolución de exercicios. Hai dúas clases de PL de 110 minutos á semana, para un total de 220 minutos á semana.

ISEP adoptou o *Learning Management System* (LMS) Moodle (*Modular Object-Oriented Dynamic Learning Environment).* Moodle serve para centralizar e poñer a disposición a información relacionada co curso, incluídos os contidos, o "Arquivo da unidade curricular (FUC)" e o horario das clases, e tamén se usa para a presentación de traballos e como canle de comunicación e interacción entre profesores e alumnos.

En APROG, os conceptos esenciais asociados á lóxica de programación baséanse no paradigma de procedemento con foco no deseño de algoritmos. Os algoritmos implementanse usando a linguaxe Java para mellorar unha transición máis rápida do paradigma de procedemento ao paradigma de programación orientado a obxectos.

APROG é o primeiro curso de desenvolvemento de *software* de LEI e é un marco para adquirir habilidades de programación de futuros enxeñeiros en informática. O desenvolvemento de *software* é unha actividade que cobra importancia con proxectos cada vez máis complexos e grandes. O desenvolvemento de proxectos adoita estar asegurado por varios equipos ao mesmo tempo, ás veces multidisciplinares e xeograficamente dispersos (Guzmán, Ramos, Seco, & Esteban, 2011). Esta realidade impón novos métodos de organización e traballo. As chamadas metodoloxías áxiles (Agile) (Moniruzzaman & Hossain, 2013) son moi empregadas, das cales Scrum é a máis empregada (Rubin, 2012) (Kuusinen, Gregory, Sharp, & Barroca, 2017). As empresas utilizan a metodoloxía Scrum para mellorar o traballo en equipo e promover un xeito de traballar produtivo, creativo e atractivo, baseado na autonomía e na responsabilidade.

Nos últimos anos, nas clases de PL presenciais, empregouse unha variante eduScrum de Scrum, adaptada á educación (Cardoso, Barroso, Castro, & Rocha, 2017). Neste modelo, os estudantes son responsables do desenvolvemento do proceso de aprendizaxe por delegación de profesores (Delhij, van Solingen, & Wijnands, 2015) nun ambiente colaborativo que promove a autonomía e a confianza en si mesmos.

Cada clase APROG ten un máximo de 20 alumnos e nos últimos anos houbo máis de 300 alumnos en 17 clases. Na clase, os estudantes resolven os exercicios dispoñibles semanalmente en Moodle e o profesor debe analizar as resolucións dos alumnos e opinar sobre o traballo realizado. Nas clases de PL, en eduScrum, os alumnos traballan en equipos de dous, polo que o profesor debe analizar o mesmo exercicio dez veces. Cada conxunto ten polo menos seis exercicios, o que significa a análise semanal de aproximadamente 60 exercicios nun período de 220 minutos, o que corresponde a un tempo inferior a catro minutos para cada exercicio.

Con esta carga de traballo non é posible realizar unha análise completa e profunda dos exercicios. A solución é unha mostra de avaliación, seleccionando só algúns exercicios, con comentarios incompletos e demasiado lentos. Neste escenario, os estudantes non poden realizar o seu traballo nin desenvolver as súas habilidades de programación o máis rápido que queiran, especialmente porque non teñen forma de validar completamente o traballo realizado. Ademais, hai unha escasa asistencia de docentes, xa que o tempo dispoñible para unha posible aclaración é moi escaso. A falta de *feedback* e seguimento dos profesores sobre o traballo dos estudantes identificouse como unha das dificultades máis importantes asociadas á aprendizaxe do código (Koulouri, Lauria, & Macredie, 2015), o que aumenta o desánimo dos estudantes.

Buscouse unha solución para reducir a carga de traballo de avaliación do profesorado, dándolle máis tempo para aclarar as dúbidas dos alumnos e, ao mesmo tempo, proporcionarlles aos estudantes mecanismos para axudar na avaliación do seu propio código. Con este fin, buscamos ferramentas que poidan apoiar o proceso para obter unha avaliación máis rápida, dando aos alumnos unha maior autonomía.

Identificáronse e analizáronse algunhas ferramentas potencialmente interesantes para o propósito proposto. Unha vez definidos algúns criterios de selección, decidiuse utilizar o VPL como soporte para a realización dun estudo.

O obxectivo do estudo foi analizar o uso potencial da VPL, en particular no que se refire á súa contribución a facer máis eficaz e autónoma a aprendizaxe dos alumnos, o seu impacto na redución da carga de traballo do profesor na avaliación. das obras e a compatibilidade do seu uso coa metodoloxía eduScrum.

METODOLOXÍA DA INVESTIGACIÓN

Ao comezo deste traballo, analizáronse os diversos pasos e condicións necesarios para a súa realización. O primeiro paso foi realizar unha revisión da literatura sobre programación docente, en particular o uso de métodos e ferramentas con potencial para racionalizar o proceso e responder ás lagoas identificadas na definición do problema. As buscas realizáronse nas bases de datos ACM, Elsevier (ScienceDirect), ERIC (EBSCO), IEEExplore, Springerlink e Web of Science e no buscador Google Scholar, así como en repositorios científicos, especialmente RCAAP e RECIPP. A investigación realizouse en inglés e portugués no segundo semestre de 2016 e considerouse publicación desde 2005. Nalgúns casos empregáronse publicacións máis antigas porque son relevantes para o tema e contribuíron ao establecemento dunha liña de evolución tecnolóxica.

Na investigación empregáronse conxuntamente os seguintes termos: clasificación automática, avaliación automática, tarefas de programación, aprendizaxe da linguaxe de programación e ensino de programación. Durante o desenvolvemento do traballo identificáronse publicacións máis recentes, algunhas delas empregadas para este traballo.

Identificáronse varias ferramentas, 35 das cales foron discutidas brevemente, principalmente en relación co seu propósito, modo de operación, linguaxes de programación compatibles e funcionalidade xeral. Para seleccionar a ferramenta a usar, definíronse algúns requisitos. A ferramenta debe ser gratuíta, segura, integrada con Moodle, fácil de usar, axeitada para a programación, permitir a carga de código (*upload*), traballar con Java e ter funcionalidade antiplaxio.

Despois desta fase, algunhas destas ferramentas analizáronse con máis detalle e escolleuse VPL, un *plugin* de Moodle. Os principais motivos desta elección foron a súa sinxeleza de uso e versatilidade, e para soportar varias linguaxes de programación (con vistas ao uso futuro noutras unidades curriculares), así como a existencia de máis literatura e material de apoio sobre este sistema que noutros analizados.

Para obter resultados que nos permitan avaliar se o uso da ferramenta elixida no contexto de APROG podería contribuír á resolución do problema identificado, deseñáronse e realizáronse experimentos durante as clases de PL. A VPL utilizouse nos cursos escolares 2017/2018 e 2018/2019 nalgunhas clases de PL seleccionadas debido á dispoñibilidade de profesores para colaborar na experiencia.

Os datos do experimento obtivéronse a través da observación directa, o rexistro automático das actividades realizadas en Moodle e principalmente a través de enquisas en liña a alumnos e profesores. A enquisa a estudantes en 2018/2019 mellorou moito en comparación con 2017/2018, aclarando algúns problemas e engadindo outros que demostraron ser importantes.

Tamén nesta segunda edición houbo varias melloras no proceso, o número de alumnos foi moito maior que no curso escolar anterior e as taxas de participación, en función do número de presentacións, foron extraordinariamente superiores. Por estes motivos, só se consideraron para a análise os datos do curso escolar 2018/2019.

DESCRICIÓN DO EXPERIMENTO

Para o uso da VPL no contexto de APROG, realizáronse algunhas reunións iniciais nas que se identificaron as necesidades materiais, técnicas e de recursos humanos.

De xuño a xullo de 2017 leváronse a cabo algúns experimentos preliminares que incluían a instalación de Moodle e VPL nunha máquina virtual dedicada e a creación de tres usuarios con tres perfís diferentes: administrador, profesor e alumno. Creouse unha actividade VPL para resolver un problema sinxelo e probouse en Python e Java.

Despois desta proba inicial exitosa, pasaron a unha fase máis administrativa para usar VPL no contexto real das clases APROG. O director de LEI e o xestor da unidade de curso de APROG foron informados e ambos aceptaron o experimento, pero houbo que cumprir algunhas condicións. As clases de APROG tiveron que proceder como de costume, como estaba previsto, para que a experiencia non tivese impacto na avaliación e deberían ser iguais para todos os estudantes, independentemente de que empregasen ou non VPL. Outra condición era que o uso de VPL non podía ser obrigatorio, xa que foi a decisión do alumno participar na experiencia. Estas condicións foron aceptadas e cumpridas polo autor do estudo e os profesores implicados no experimento.

Para levar a cabo o experimento foi necesario instalar o *plugin* VPL en Moodle. Solicitouse a autorización á Presidencia do ISEP, que foi concedida. Despois, o *plugin* instalouse no Moodle institucional de ISEP, e foi realizado polo técnico de ISEP responsable da administración de Moodle e foi supervisado e supervisado polo autor do estudo. Tamén se instalaron compiladores para as linguaxes de programación destinadas a usarse coa VPL e comprobáronse as condicións de acceso remoto, os permisos de acceso e as configuracións de rede e *firewall*. Despois da instalación, realizáronse probas funcionais e corrixíronse as deficiencias atopadas para garantir o rendemento do sistema.

Unha vez cumpridas as condicións técnicas, loxísticas, pedagóxicas e xurídicas, a fase de implementación do experimento comezou cunha reflexión sobre os exercicios APROG que se empregarán na VPL. Débese reducir o número de exercicios para non interferir no proceso de ensino-aprendizaxe en APROG. Polo tanto, escolléronse algúns exercicios en función da súa importancia e facilidade de adaptación á VPL sen distorsionar o contexto e o seu propósito. Elixíronse seis exercicios de programación, dous en cada unha das tres semanas nas que se levaría a cabo o experimento, que abordou o uso básico dos principais elementos de programación.

Decidiuse usar VPL a partir da segunda semana dedicada á codificación Java para permitir aos estudantes, na primeira semana, o contacto inicial co IDE e a sintaxe do Java, sen introducir unha nova ferramenta. Isto faría máis doado para os estudantes iniciar o contacto cun novo contexto, coa vantaxe de que inicialmente codifican no IDE e só despois cargan o código no VPL, permitindo un axuste máis suave.

Para implementar as actividades en Moodle, os seis exercicios seleccionados foron analizados e adaptados á VPL. O funcionamento do VPL baséase na análise e avaliación de casos de proba e o resultado determínase en función da similitude das cadeas entre o resultado obtido e o resultado esperado. Por iso, foi necesario deseñar e implementar casos de proba que cubrisen as distintas posibilidades de saída, para permitir a avaliación das obras.

Como se mencionou, o experimento realizouse no curso escolar 2017/2018 e repetiuse en 2018/2019, cun proceso mellorado e que implicou máis clases, o que significou máis alumnos e presentacións. Entre estes dous cursos académicos fixéronse diversos axustes e desenvolvéronse novas características que repercutiron nos plans técnicos e pedagóxicos. Cambiáronse os procesos de verificación da presentación, o que permitiu unha maior flexibilidade para o alumno e evitou algúns problemas verificados na edición 2017/2018, a saber, en relación ao ficheiro a enviar que xa non ten un nome obrigatorio. Tamén se implementaron funcións de análise de estilo de código, baseadas na observación feita polo autor do estudo e nas opinións doutros profesores que participaron no experimento.

Esta característica desenvolveuse completamente desde cero, pero dun xeito compatible coa estrutura de ficheiros VPL preexistente. É unha opción flexible porque non é obrigatoria e permite ao profesor establecer parámetros segundo o que queira avaliar. Tamén se implementou un sistema de xestión de versións para esta función, que permite o almacenamento na nube e unha xestión centralizada e flexible, que lle permite evolucionar.

Estas funcións utilizáronse no curso escolar 2018/2019, engadindo a análise do proceso de codificación á verificación dos resultados que normalmente realiza a VPL.

O plan de investigación non estaba previsto para o curso 2019/2020, pero debido á experiencia de anos anteriores e á boa aceptación dos estudantes, o autor decidiu empregar o VPL nesta edición de APROG, contactando co novo xerente do Unidade curricular que dixo saber do estudo realizado en anos anteriores. Por este motivo, o novo xerente planeaba usar unha ferramenta de avaliación de código de xeito sistemático para todos os exercicios e para todos os estudantes. Non obstante, dado que descoñecía o funcionamento do VPL e xa utilizara Mooshak en varias competicións de programación, tiña a intención de optar por isto pero aceptou usar o VPL despois das presentacións de Mooshak e de forma complementaria. Polo tanto, tamén neste curso escolar empregouse o VPL en APROG. Realizouse unha enquisa na área de APROG UC Moodle, potencialmente interesante para o alcance deste traballo, pero os resultados non se presentan aquí xa que non se coñecen oportunamente.

RESULTADOS E CONCLUSIÓNS

Os principais resultados deste estudo obtivéronse principalmente a partir de datos de enquisas realizadas con alumnos. As enquisas realizáronse nos cursos 2017/2018 e 2018/2019, pero só se consideraron as respostas do curso 2018/2019 do estudo, xa que ese ano as enquisas ampliáronse e melloráronse e foron respondidas por un número moito maior de estudantes.

A partir da análise xeral dos resultados, obsérvase que os estudantes amosan unha gran predisposición para o uso das tecnoloxías, así como a posibilidade de empregar ferramentas que permitan o estudo autónomo.

Os alumnos non revelaron dificultades significativas para usar o VPL, mostrando gran interese e entusiasmo durante a experiencia, como se pode concluír nas respostas á enquisa e tamén no que observaron os profesores durante as clases de APROG PL.

En canto ao proceso de ensino-aprendizaxe, a apreciación positiva xeral dos estudantes é evidente e os resultados son moi gratificantes polo esforzo e compromiso dedicado ao estudo e os motivadores para o futuro uso da VPL no ensino da programación.

Tamén da análise das respostas ás enquisas pódese concluír que a preparación do experimento foi axeitada, proporcionando aos alumnos as condicións necesarias para o desenvolvemento das actividades.

Como un dos obxectivos deste traballo é reducir o tempo empregado polos profesores na avaliación, tamén se realizaron investigacións cos profesores e a súa opinión foi moi favorable ao uso de VPL.

Como parte deste traballo, publicáronse artigos científicos en conferencias con revisión e presentáronse publicamente en eventos internacionais, demostrando a súa validez e aceptación pola comunidade científica. Da lista de artigos, debe destacarse o artigo publicado nunha revista Impact Factor (*JCR® Impact Factor*) medido por ISI Thomson, unha revista situada no segundo cuartil.

Realizáronse outras actividades de divulgación para este estudo e sobre o uso da VPL, en particular en eventos con presenza de profesores de educación secundaria e superior.

O autor é membro do equipo portugués do proxecto europeo Up2U (*Up to University*), que ten como obxectivo reducir a brecha entre a educación secundaria e a superior fomentando o uso de tecnoloxía e metodoloxía que presuntamente atoparán os estudantes na educación superior. Neste contexto, implementouse un ecosistema baseado en Moodle e púxose a disposición das escolas secundarias europeas de varios países. O autor foi invitado a instalar o *plugin* VPL neste ecosistema e actualmente está dispoñible a creación de actividades VPL.

Tamén dentro do ámbito deste traballo creouse un curso de programación Java apoiado por ferramentas de avaliación automática, nomeadamente o VPL e Mooshak. O curso está dirixido a profesores e foi acreditado polo Consello Científico-Pedagóxico de Formación Continua, a entidade nacional portuguesa para esta área.

En resumo, tendo en conta o traballo realizado de acordo co plan de investigación e os resultados e achegas acadados, pódese considerar que os obxectivos inicialmente propostos para este traballo foron alcanzados globalmente.

TABLE OF CONTENTS

INDEX OF FIGURES

INDEX OF TABLES

INDEX OF CHARTS

INDEX OF ANNEXES

ACRONYMS

This document uses acronyms and terms of common names, justified both by their frequent use throughout the document and by their use in this field of knowledge, each of which is presented in its first use. The list of acronyms is presented below, in alphabetical order.

ACM	Association for Computing Machinery
AI	Artificial Intelligence
ANSI	American National Standards Institute
APB	*Associação Portuguesa de Bancos* (Portuguese Bank Association)
API	Application Programming Interface
APROG	Algorithmics and programming
AR	Augmented Reality
ASD	Adaptive Software Development
ASME	American Society of Mechanical Engineers
AT&T	American Telephone and Telegraph
AVA	*Ambientes Virtuais de Aprendizagem* (Virtual Learning Environments)
B2B	Business to Business
B2C	Business to Consumer
BASIC	Beginner's All-purpose Symbolic Instruction Code
BDDAD	*Bases de Dados* (Databases)
b-Learning	blended learning
BNF	Backus Naur Form
BOSS	Bob’s Online Submission System
CEDETEL	*Centro de Desenvolvimento de Telecomunicações em Castela e Leão* (Telecommunications Development Centre in Castilla and León)
COBOL	COmmon Business Oriented Language
COOL	C-like Object Oriented Language
CPS	Cyber Physical Systems
CU	Curricular Unit
DEI	*Departamento de Engenharia Informática* (IT Engineering Department)
DevOps	Development and Operations
D-Learning	Distance Learning
DNS	Domain Name System
DSDM	Dynamic Systems Development Method
ECMA	European Computer Manufacturers Association
e-Commerce	electronic commerce

e-Learning	electronic learning
ENIAC	Electronic Numerical Integrator and Calculator
FAQ	Frequently Asked Questions
FARE	*Foco, Análise, Resolução, Execução* (Focus, Analysis, Resolution, Execution)
FDD	Feature Driven Development
FUC	*Ficha de Unidade Curricular* (Curricular Unit Sheet)
GNU	GNU is Not Unix
GPL	General Public Licenses
GUI	Graphical User Interface
HoGG	Homework Generation and Grading
HTML	Hyper Text Markup Language
IaaS	Infrastructure as a Service
IAL	International Algebraic Language
IBM	International Business Machines
ICPC	International Collegiate Programming Contest
ICT	Information and Communication Technologies
IDE	Integrated Development Environment
IDEAL	Identification, Definition, Exploration, Action, Learn
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineering
IFB	*Instituto de Formação Bancária* (Banking Training Institute)
IOI	International Olympiad in Informatics
IoT	Internet of Things
ISEP	*Instituto Superior de Engenharia do Porto*
ISO	International Organization for Standardization
JDK	Java Development Kit
JIT	just-in-time
JSP	Java Server Pages
JVM	Java Virtual Machine
LAN	local area network
LCMS	Learning Content Management System
LEI	*Licenciatura em Engenharia Informática* (Degree in Computer Engineering)
LMS	Learning Management System
MIT	Massachusetts Institute of Technology
m-Learning	mobile learning
Moodle	Modular Object-Oriented Dynamic Learning Environment
MRI	Matz Ruby Implementation
OECD	Organisation for Economic Cooperation and Development

OKI	Open Knowledge Initiative
OO	Object Oriented
OOPSLA	Object-Oriented Programming, Systems, Languages and Applications
OU	The Open University
PaaS	Platform as a Service
PC2	Programming Contest Control
PETCHA	Programming Exercises TeaCHing Assistant
PHP	Hypertext Preprocessor
PHP-FIG	PHP Framework Interoperability Group
PISA	Programme for International Student Assessment
PPROG	*Paradigmas da Programação* (Programming Paradigms)
PROLOG	PROgrammation en LOGique
PSRs	PHP Standards Recommendations
PYPL	PopularitY of Programming Language Index
RAD	Rapid Application Development
RCAAP	*Repositórios Científicos de Acesso Aberto de Portugal* (Open Access Scientific Repositories of Portugal)
RECIPP	*Repositório Científico do Instituto Politécnico do Porto* (Scientific Repository of the Polytechnic Institute of Porto)
RPC	Remote Procedure Call
RUC	*Responsável de Unidade Curricular* (Head of the Curricular Unit)
SaaS	Software as a Service
SAC	System for Automated Assistance in Correction of Programming Exercises
SCP	Smart Connected Products
SDK	Software Development Kit
SSL	Secure Sockets Layer
SWERC	South Western Europe Regional ACM Programming Contest
u-Learning	ubiquitous learning
UNED	*Universidad Nacional de Educación a Distancia*
UNESCO	United Nations Educational, Scientific and Cultural Organization
VLSI	Very large-scale integration
VNC	Virtual Network Computing
VPL	Virtual Programming Lab
VPN	Virtual Private Network
VR	Virtual Reality
Xinetd	Extended Internet Services Daemon
XP	Extreme Programming

1 INTRODUCTION

"It is a lousy idea, one that you cannot change."

Montaigne

In this first chapter, the general contextualization of the environment in which the research is carried out is made and the problem identified is presented. The proposed objectives are also mentioned, as well as the contributions expected to result from this work.

Next, the research methodology used and the author's personal and institutional motivations for this work are mentioned.

The chapter ends with a presentation of the organisational structure of the thesis and a brief description of each of the chapters constituting it.

1.1 BACKGROUND

In recent years, we have witnessed an exponential growth in computer technology. This accelerated evolution has caused profound changes in our lives, in social relations and in the most diverse economic activities. The way we relate, communicate, travel, buy and sell today is radically different from what it was just a few decades ago and, in the overwhelming majority of cases, depends on Information and Communication Technologies (ICT).

According to Fiolhais (Fiolhais, 2005), information technology has played an overwhelming role in the development of modern society. The author points out that computers are everywhere today, and we could hardly imagine our life without them. This statement of almost a decade and a half ago, so far from today's reality is, however, as valid today as it was then. Even without realizing it, almost all of our actions are somehow associated with information technology. The mere fact that we turn on a switch, will cause a system somewhere to receive the information that we are consuming energy. Energy meters themselves are now, in many situations, digital and some with remote metering (smart meters) connected directly to a data centre. The power plants, the electricity transmission and distribution networks and all related infrastructures and systems are managed electronically (Paiva, 2015). But there are many more examples of the use of information technology, such as water distribution network management systems, air traffic control, the stock exchange, commercial and financial transactions, communications, medicine, military, surveillance, education, among many others.

To this day, we have come a long way. The genesis of computers has millennial roots. It is believed that the oldest tool related to calculus is the abacus, an instrument invented about 5 000 years ago, which has been perfected over time and was the precursor of the calculating machine (Williams, 1985). More recently, in the 17th century, Pascal's machine marked an era in technological evolution. Nonetheless, the ancestral machine closest to the concept of modern computers was conceived, although not realized, by Charles Babbage, still in the first half of the 19th century. After Babbage, new developments took place with Hollerith's perforated cards in particular, who founded the company that would give rise to International Business Machines (IBM) (Kistermann, 2005). Another important milestone was the design of the Turing machine, developed by Alan Turing, an English mathematician, founder of Computer Science and percussionist of Artificial Intelligence (AI), and whose concepts were used in the construction of equipment to decipher German army codes during World War II (Rojas & Hashagen, 2000).

With the development of electric valves, Electronic Numerical Integrator and Calculator (ENIAC) emerged in the 40's of the 20th century and was considered the first general-purpose electronic computer, in what was a gigantic leap from the mechanical devices used up to then. With the appearance of the transistor a new revolution takes place with a radical decrease in the volume, cost and weight of equipment, and a huge increase in processing speed. This period became known as the second generation (Rojas & Hashagen, 2000).

The next stage was characterised by a trend towards miniaturisation of devices having become a precursor to the development of integrated circuits. Processing speed increased by dozens of times over the previous stage as well as there being a large increase in storage capacity based on semiconductor-built memories.

However, the most striking period in the history of computers would have been as of the mid 1970s of the 20th with the emergence of Very large-scale integration (VLSI) technology, which resulted in processors with good performances, which boosted the development of small computers at affordable costs and gave rise to a new concept: the personal computer. It is also in this period that Microsoft and Apple appear, two colossuses that are present all over the world today. Also, in programming languages, operating systems and software, there were major developments such as the launch of the Windows operating system or the emergence of Linux.

Currently, designations such as Artificial Intelligence, neural networks, telemedicine, Virtual Reality (VR), Augmented Reality (AR), cloud computing or the internet of things, usually referred to as Internet of Things (IoT) are already widely used (Farshida, Paschena, Erikssonb, & Kietzmann, 2018) (Schmidhuber, 2015) (Atzori, Morabito, & Iera, 2010). This accelerated technological evolution, unprecedented in human history, was predicted in 1965 by Gordon Moore (Moore G. E., 1965). The later called Moore's law advocated that every two years, on average, there would be a doubling of the number of components per integrated circuit. This is a concept which remains valid to this day.

Given that computer equipment (hardware) is essential for computing, there is, nevertheless, another indispensable part to the functioning of the systems: the software. Thus, the development of software is a crucial activity for the functioning of society as a whole. Knowing how to program is an emerging competence in the 21st century (Engelhardt & Balanskat, 2015).

Producing software is not just about programming, but it is a key task for the success of the process. In this context, there is a need for new programmers and software engineers in order to ensure the operation and development of computer systems, so teaching how to program is essential for the development of new applications, technological evolution and the sustainability of society as we know it.

Being that teaching programming is a complex and challenging task for any teacher, especially when teaching beginner programmers, it is not an easy task for most students either (Moström, 2011). According to Anabela Gomes (Gomes A. , 2010), this is demonstrated in the typically high failure rates in the introductory programming courses, all over the world and in all degrees and education systems. The search for new methods, strategies, means, tools and technologies that can improve the process is thus a constant. Proof of this is the number of publications and research work on the subject (Bennedsen, Caspersen, & Kölling, 2008) (Luxton-Reilly, et al., 2018), with research in the field of teaching, but also in the technological areas and even in psychology, particularly with regard to motivation (Lykke, Coto, Jantzen, Mora, & Vandel, 2015) (Tsukamoto, Nagumo, Takemura, & Nitta, 2012). There are no magic formulas to turn a person into a competent programmer, but there will surely be opportunities to improve the teaching-learning process of programming by enhancing and developing students' skills, making the process less painful and more effective.

1.2 THE PROBLEM

In the course of Computer Engineering (LEI) of the Department of Computer Engineering (DEI) of the Polytechnic Institute of Porto (ISEP) students are taught, in the first semester of the first year, the curricular unit (CU) of Algorithmics and Programming (APROG). This CU aims to provide students with an essential skill for a future computer engineer: programming.

Here, the fundamental concepts associated with the programming logic based on the procedural programming paradigm are introduced, with special focus on modelling and algorithmic problem solving. The implementation of the designed algorithms is carried out using the Java language in order to foster a faster transition from the procedural paradigm to the object-oriented programming paradigm (in later curricular units).

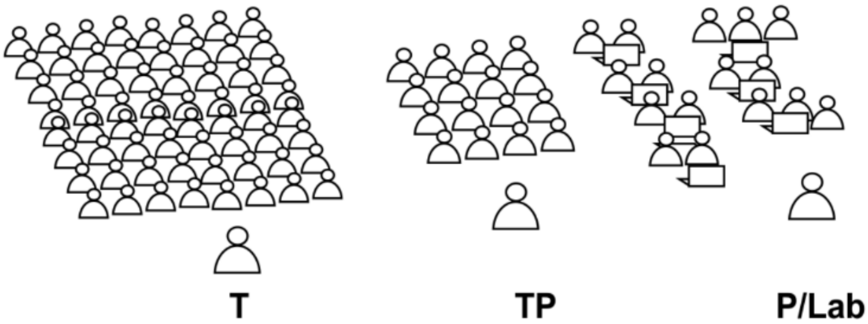


Figure 1 – APROG lesson types

In APROG, the weekly load is divided into three types of classes: theoretical (T), theoretical-practical (TP) and laboratory-practical (PL), as presented in Figure 1.

Each of these class types has a different weekly schedule, as shown in Table1.

Table1 – Weekly workload by type of class

Type of class	APROG		
	Number of lessons per week	*Lesson length (min.)*	*Total weekly lesson length (min.)*
Theoretical (T)	1	50	50
Theoretical-practical (TP)	1	50	50
Laboratorial practice (PL)	2	110	220

In recent years, the eduScrum (Delhij, van Solingen, & Wijnands, 2015) methodology has been applied in face-to-face lessons, where students organize themselves into teams. This methodology is a variant of Scrum applied to education, in which students are responsible for the learning process by the teacher's delegation. Scrum (Sutherland, 2014) is suitable for complex product development and is widely used in the production and maintenance of software, in an agile, iterative and incremental process. Companies use the Scrum methodology to enhance teamwork and promote a productive, creative and attractive way of working, based on autonomy and responsibility.

EduScrum advocates that a team should consist of a teacher (Product Owner) and four students. However, in this case, as it is an introductory course, for logistical and operational reasons, the teams only include two students (exceptionally three, if there is an odd number of students). The work of each team is based on autonomy, transparency, self-organization, collaboration, multidisciplinarity and responsibility.

At ISEP, Moodle was adopted as Learning Management System (LMS), thus also being used in APROG. Its main purpose is to centralize all information related to the CU and to serve as a means of communication and interaction between teachers and students. In Moodle, the contents, the Curriculum Unit Form (FUC) and the schedule of the classes for the whole school term are made available, and it is also used for the submission of assignments and presentation of the respective individual evaluation.

For each week of classes, a set of exercises is made available which the students must solve. In each class, the teacher should analyse the students' responses and give feedback on the work developed. Considering that each class can have about twenty students and that each worksheet can have six or more exercises, the teacher will need to review dozens of assignments and give feedback each week. That is one of the problems that have been identified.

Given the number of exercises solved by students and the need for teachers to assess them and considering other assessment to be made during the school term, the assessment task is very intense and extremely time-consuming. With a large set of exercises, it is (almost) impossible to thoroughly evaluate them all, with the solution being to evaluate by sampling. This means that the feedback given to students on their own resolutions is incomplete and (almost always) delayed in time.

In this scenario, it is observed that students cannot do their work or develop their programming skills as quickly as desirable, mainly because they have no way to fully validate the work they have done. Moreover, there is a lack of support from the teacher, as the time available for any clarification is very short. Thus, the deficiency of feedback and of monitoring by the teacher in relation to student's work was identified as one of the most important difficulties associated with learning programming (Koulouri, Lauria, & Macredie, 2015), increasing the students' discouragement.

Knowing that evaluating and providing significant feedback is a popular and effective method of engaging students in programming (Verdú, et al., 2012) to solve this problem effectively and quickly, teachers suggested that classes should have fewer students or that less exercise and less evaluation time should be considered. However, for logistical and financial reasons, it is not possible to reduce the number of students per class. Regarding the number of exercises to be solved, the teaching team understood that it should not be reduced, as practice is fundamental for the development of programming skills (Tavares P. C., 2018).

Therefore, a solution was sought to solve the teacher's overload of evaluation work, giving the teacher more time to clarify the student's doubts and, at the same time, empowering students with mechanisms to help them evaluate their own code. With this objective in mind we looked for tools that could support the process to obtain a faster evaluation, making the students less dependent on the presence of the teacher.

1.3 OBJECTIVES AND EXPECTED OUTCOMES

This study aims to investigate the potential of integrating a virtual learning laboratory, VPL, in the teaching of programming. For this purpose, we will use as support the APROG CU of the Degree in Computer Engineering, taught at the Department of Computer Engineering of the Instituto Superior de Engenharia do Porto, Portugal.

This CU is taught in the first half of the first year and is instrumental in developing the programming skills of future computer engineers, as it conveys the fundamental knowledge that can be applied in any programming language. In APROG, besides the teaching of algorithm and transmission of programming logic knowledge, application knowledge is also transmitted in a specific language, in this case Java. This CU has been taught in recent years using the eduScrum methodology and with support in Moodle.

Bearing in mind that the CU has many students enrolled (more than 300 per year) and that each PL class has about 20 students, an excess of time has been noted in recent years in relation to the workload of assessments by the teachers of the practical classes. Despite the considerable time devoted to evaluation, it proved insufficient to fully and comprehensively analyse all the work carried out and to provide timely feedback. Moreover, due to the considerable amount of time spent on evaluations, it is not possible to support students effectively, particularly as regards to clarifying doubts.

In this sense, we carried out an analysis aimed at implementing an automatic validation process based on VPL. It is expected that this process will reduce the time spent by teachers on evaluation, freeing them to teach and to clarify doubts expressed by students.

We intend to verify if the use of VPL has potential in order for the student to learn in a more autonomous way and for periods other than those of the face-to-face classes, since they will be able to submit their work and receive an automatic evaluation and an immediate feedback from the system.

The experience of using VPL occurred, in an initial and embryonic phase, in some classes of APROG, during the first half of the school year of 2017/2018, from September to December 2017. In the school year of 2018/2019, it was repeated after certain changes and with a significant increase in the number of students involved. Also, in the school year of 2019/2020 the VPL was used, but in a complementary perspective with respect to another tool that was used.

Thus, in addition to the general objective of analysing the potential use of the VPL, it was intended:

- To verify the feasibility and usefulness of using VPL in the teaching of curricular units in higher education;
- To check the compatibility of the use of the VPL with the eduScrum methodology;
- To verify whether, with the VPL, it is possible to make the learning of students more effective and autonomous;
- To investigate the impact of the use of VPL in reducing the teacher's workload in the evaluation process;
- To analyse whether the use of VPL can contribute to a faster and fairer evaluation by standardising evaluation and the use of anti-plagiarism tools;
- To justify the extension of the use of VPL, or any similar tool with the same purpose, to all students of APROG, as well as to other curricular units of programming and/or other courses.

It is hoped that this study can prove that the integration of VPL in the teaching of programming languages can result in a more agile process, with increased student motivation and improved overall results in teaching programming.

1.4 Research Methodology

Research is the collection, analysis and interpretation of data in order to understand and characterise a specific phenomenon. For this, it is necessary to establish planning and to set tasks and objectives. It is also essential to analyse, evaluate and, where necessary and possible, create the conditions for carrying out the planned activities. Should there be insurmountable constraints, it is also necessary to redefine and adapt the plan so as to make it feasible, without foregoing trying to achieve the objectives previously established.

Within this context, a literature review was carried out on the teaching of programming, with a special focus on the use of methods and tools with the potential to streamline the process and fill the gaps identified in the problem definition.

From the analysis carried out, the VPL was identified as a versatile tool, free of charge, compatible with the technical and logistical conditions available to us and potentially effective for the intended purpose. Thus, since the VPL is a plugin from Moodle, a pilot was setup on a local machine to verify its functionality and usefulness.

After this successful experiment, a new literature review on the VPL was carried out and it was decided to proceed with the experiment in a real context. Accordingly, a previous personal approach was made with the course director and the head of the curriculum unit (RUC). Once both had given their consent, formal authorisations to conduct the experiment were requested:

- To the head of the APROG CU, for potential pedagogical implications;
- To the Director of the LEI course, as it could affect issues of assessment and coordination with other course units;
- To the Presidency of ISEP, in the technical and logistic component, for the need to install the VPL plugin in the institutional Moodle.

The intention of the experiment was also presented to the APROG teaching team, to increase critical mass, garner opinions and enrich the process, as well as to extend the experiment to a significant number of classes.

After installing the VPL, some functional tests were carried out and some technical difficulties were identified, which have since been overcome.

We analysed the exercises usually used in APROG from which some were chosen for use in the VPL. Various tests were designed and implemented according to each sprint of the eduScrum process.

After the preparation stage, we moved on to the experience that took place in PL classes for three consecutive weeks.

The experience took place in the 2017/2018 school year, and was repeated in the 2018/2019 school year, with more students and some improvements in the process.

Student opinion surveys on the use of VPL were carried out (in both editions) and are attached to this document.

For the purposes of analysis, only data for the 2018/2019 school year is used because of a higher number of students involved compared to the previous school year and because participation rates, based on the number of submissions, were significantly higher.

The teachers who participated in the experiment were also questioned about the use and usefulness of the VPL.

1.5 MOTIVATION

The decision to carry out this work and the motivation to do so are primarily the responsibility of (and the addressees of) the students. It is fundamentally because of them and for them that this work has been undertaken, intending to contribute to the improvement of the teaching-learning process of programming.

1.5.1 Personal Motivation

Neither computer science nor the pedagogy being the background training of the author, it will be licit to question what will drive an electro technician to teach algorithmics. The truth is that the author has a great passion for teaching, for teaching and learning, and for being in contact with students, regardless of their age, the subjects to be dealt with or the level or context of the training. In addition, more than three decades ago he began his professional activity in computer programming, having only later started his teaching activity on a regular basis. Initially, he taught at a different level of education and other subjects and for about twelve years in the area of information technology, more specifically, linked to the teaching of programming.

Getting someone to learn how to program is an even more special experience than teaching any other subject. It is a challenging, difficult and demanding process, but when students begin to make their first implementations and achieve results it is very motivating and rewarding. Programming, in addition to the concepts and syntax of language, requires thinking, interpreting and designing solutions.

As a rule, students who are new to programming are starting in higher education for the first time. This change of environment, context and paradigm regarding the teaching and learning models between secondary and higher education represents an even greater challenge, and it is up to the teacher to help minimize this impact.

Given the rapid technological evolution and the enormous volume of scientific production, the need for updating in order to evolve personally and professionally is clear. The expectation was that this work would enable the study and deepening of the themes related to the teaching of programming, which was absolutely confirmed.

1.5.2 Institutional Motivation

ISEP, the institution where the author teaches, was founded in 1852 and enjoys great prestige at a national and international level, with IT being one of its most important and recognized areas. The Computer Engineering Department (DEI) counts on many hundreds of students and several dozen teachers, many of them highly qualified, and several with great merit nationally and internationally acknowledged. Over the years, DEI has taken action to innovate, improve and implement good engineering practices, qualifying its trainees for outstanding professional performance.

The author, as a DEI lecturer, also intends to contribute to increasing the quality of teaching and the prestige of the department. This prestige is the result of a great effort made in recent years, which has catalyzed the name of ISEP and DEI, with great impact on the appeal of students and, even more importantly, on the great employability of our graduates.

Obtaining a doctoral degree, besides obviously representing an added personal and professional value for the author, will also contribute to the scientific strengthening of the institution, helping to increase the number of doctorates and thus helping to meet mandatory ratios for the functioning of the courses.

This research is expected to contribute to pedagogical innovation and to the improvement of teaching methods and processes, so that the teaching of programming is more efficient and effective. It is also intended to divulge and disseminate the use of automatic validation tools in order to increase the quantity and quality of software tests with a view to helping students and teachers.

1.6 ORGANISATION OF THE DOCUMENT

This document is organised in seven chapters, and contains eleven annexes, which present the main results, contributions and conclusions of this doctoral work.

In this chapter we proceed to the contextualization, making a brief description of the framework of this research project. The problem which is intended to contribute to is then identified and characterised. The research methodology is also described and the reasons for this work are explained.

The end of this chapter covers this section describing how the document is organised.

Chapter 2 reviews the development process of software and its importance in the context of a society that makes massive use of computers. Its use, usefulness and risks are characterised and specific issues of each of the areas addressed are also mentioned.

Technical aspects such as the different languages and programming, their characteristics, evolution and fields of application, as well as integrated development environments (IDE) are listed.

They also refer to ways of organising the development processes of software, in particular with regard to agile methodologies.

Chapter 3 is dedicated to the teaching-learning process of computer programming. It addresses issues such as problem solving, teaching programming and its challenges, and the basis of programming: algorithms.

Pedagogical aspects related to face-to-face learning and blended learning are also analysed, with the final section of the chapter being dedicated to LMS, its origins, characteristics and functionalities, and some examples are also presented, with Moodle being highlighted.

In chapter 4 an analysis is made of the support mechanisms for teaching programming, with emphasis on automatic code evaluation, and a list of examples of tools for this purpose is presented. A detailed analysis of some of these tools is made and the reasons for choosing the VPL as a study support tool are presented.

The description of the experience is made in Chapter 5, where the context of the experience support unit and the whole process of experience analysis, design, preparation and implementation are presented in detail.

All the actions carried out in each of the editions are detailed, as well as the changes made to improve the process, highlighting the inclusion of some additional checks regarding the codification system.

Chapter 6 is devoted to the presentation and analysis of the results of the study, referring to the data collection mechanisms and processes used.

The results and the resulting information are presented for analysis and interpretation.

At the end of the chapter the publications made during this work as well as other results and actions within the scope of the same are presented.

Finally, chapter 7 presents the conclusions of the work carried out and envisages future improvements and possible lines of research for possible future work.

2 THE IMPORTANCE OF SOFTWARE DEVELOPMENT

"The evolution of man necessarily involves the search for knowledge."

Sun Tzu

In this chapter, a framework is made of information technology and computers in society, giving examples in different areas such as health, education and the economy. A reflection on programming languages is presented, highlighting their evolution and the different types of languages and fields of application.

Data are also presented on technological developments, integrated development environments for code writing and new device and application trends.

Finally, reference is made to the ways in which the development processes of software are organized, with a special focus on agile methodologies.

2.1 INFORMATION TECHNOLOGY AND SOCIETY

The evolution of science and technology has a direct impact on our daily lives and the way we communicate, travel, study, work and even on how we relate to one another. Many of society's radical and structural changes are directly associated with technology, which has empowered means of communication and mobility unthinkable only a few centuries ago.

Today's society is highly dependent on technology, and, in the Western world, it is unimaginable to live without electricity, telecommunications or the Internet. Technology in general and information technology in particular is found in all areas of activity and sometimes in the most unexpected and unsuspected process. Even if this is not always evident, any action we take will depend, in some way, on information technology. From health to business, from transport to education, from sport to information, in everything a strong presence of IT means and processes can be found. The military area, too, has an increasingly strong computer component and, over time, has made major contributions to technical and scientific advances.

Computers have significantly altered some of the processes we regularly perform. From the way we deal with the tax authorities, social security and other public agencies, to renewing documents or obtaining information, to commercial transactions, the relationship with banks or insurance companies, nowadays, everything can be done anytime and anywhere, at a click's distance.

The advantages of the use of computerised means are indisputable. The increase in data calculation and storage capacity, together with the possibilities for parallel processing, have allowed enormous scientific advances to be made. However, negative aspects of technology are increasingly discussed, such as new health problems resulting from its use, the "dehumanisation" of relationships, or computer piracy and sabotage. In any case, information technology is a reality and, if used well, of enormous use to society.

2.1.1 The health sector

The health sector is an excellent field of application for information technology and computers. Here we will be able to identify its use for various purposes such as the administrative and technical management of a hospital or to perform complementary diagnostic tests. Many medical devices have processors and other components typical of computers that give them a clearly computerized context, working with pre-defined or programmable software depending on their complexity and function. Medical imaging is an area where images of the human body are created for medical purposes, and many of the scanning and imaging techniques used there are based on computer technology. For example, in an axial-computer tomography, digital geometry processing techniques are used to obtain three-dimensional images.

Moreover, in the study and recording of historical data on diseases or in the analysis of cellular structures and micro-organisms, computers have proved to be a precious ally, enabling tremendous developments in the medical and pharmaceutical fields.

Another crucial area is decision making, which can be strongly supported by computer systems enhancing the quality and efficiency of health care delivery and reducing error (Jardim, 2013). Patients' clinical records in digital format, with access to their entire personal and family history, diagnosed diseases, prescribed medication and test results, accessible at anytime and anywhere, allow all medical processes to be streamlined with clear advantages for everyone. However, as in other areas, access to this sensitive information can be a vulnerability aspect that needs to be taken into account and prevented.

The amount and diversity of data in medical records, management system, database of doctors and other personnel, as well as other systems, represent major challenges with respect to interoperability, quality, safety and reliability conditions, as well as data storage, processing and integrity (Plazzotta, Luna, & Quirós, 2015). Often the (several) existing applications (each for its own area and purpose) are not able to be interconnected and to communicate with each other, which represents a significant barrier to using existing data and resources for effective and integrated management.

With the growing average life expectancy of the world's population, which is expected to rise from 71 years in 2010 to 83 years by the end of the 21st century (United Nations, 2017a) (United Nations, 2017b), and by 90 years in the more developed regions, it is expected that medical computing will become even more relevant and important in the coming decades. Vital function monitoring, alarm and, in some cases, diagnostic and first intervention devices, as well as applications developed specifically for this purpose for mobile devices are already a tested and widely used reality, but they remain a fertile field of research for new solutions.

All this development is boosted by technological advances in communications, computer networks and the Internet giving rise to the concept of telemedicine (Lustig, 2012). Today it is possible to take medicine to distant places, allowing doctor-patient interaction at a distance, with the possibility of making diagnoses and sometimes surgical procedures. Today, with the Internet, it is extremely simple to observe, send or receive images or test results with those involved in different physical locations.

There are also new fields of application in the area of health informatics such as AI and machine learning (Deo, 2015). The increase in the size and complexity of the problems to be solved computationally has been accompanied by a remarkable development of storage resources and information processing management. This progress has allowed to extend the scope of use of computers to new areas and applications, increasingly demanding and complex and with promising results. Computers are being used more and more with self-learning functionalities, with large volume and complex data analysis capabilities, with more elaborate and robust algorithms, opening new possibilities in several areas.

2.1.2 The education sector

Contrary to what one might think, the use of computers in teaching is not a recent phenomenon. Ralston (Ralston & Meek, 1976) states that as early as 1958 a computer was used as a teaching machine at the Coordinated Science Laboratory[1] of the University of Illinois. This experiment was an attempt to apply the teaching machine of Skinner (Skinner, 1958), an American psychologist adept at radical behaviourism who advocated the importance of immediate feedback to the correct response. However, 60 years ago, the means available and the perspective would have been very different from today's, basically serving to store information and make it available to the student.

Since those times, we have seen incredible technological advances, with the increase in the processing and storage capacity of computers and the decrease in their dimensions, with the appearance of laptops, tablets and mobile phones and with the generalization of Internet access. This is an extraordinarily different reality, with access to a panoply of equipment and resources in teaching that enhance richer and more challenging environments for students and teachers (McKnight, et al., 2016). All this translates into a new paradigm with major implications for the organization of the school and the role of the teacher (Cairns & Malloch, 2017). As Bates points out (Bates, 2015), changes in the economy and technology require teaching with a new approach, requiring teachers to adapt their role in the classroom.

Technology must be used in favour of teaching, meeting the expectations of *digital native* students. It should be noted that, as argued by Bullen (Bullen, Morgan, & Qayyum, 2011), there are those who advocate talking about *digital learners* and not *digital natives*. This statement stems from the fact that it is not always the youngest students, who are supposed to be most familiar with the technology, who best master and use it. It is therefore crucial to involve all students in digital literacy in order to make the most of computer resources for learning.

To this end, strategies based on playful resources are increasingly used, focusing on interactivity and on trying to make the activities more attractive and closer to the students' family contexts. From here comes the concept known as gamification (Piteira & Costa, 2017) (from the English term gamification), but, in reality, it should be called gameplay, which translates into the application of concepts, mechanics and elements of games in other contexts, namely in education and teaching. This principle aims to make technology more attractive, enhancing pleasure and increasing interest, involvement and motivation in order to induce behaviours that lead to better performance (Tu, Sujo-Montes, & Yen, 2015).

Another advantage of the use of technology is that it enhances approximation, mitigating the notion of physical space by using simulators as well as remote and virtual laboratories (Heradio, de la Torre, & Dormido, 2016). These are means which, in addition to allowing for experiments and training at reduced costs (Viegas, et al., 2017), provide more attractive and enriching experiences while offering other relevant advantages (Gravier, Fayolle, Ates, & Lardon, 2008) of which the following stand out:

- Accessibility - usable by people with disabilities;
- Control - they are more easily monitored and controlled;
- Availability - they can be used at any time and from anywhere;
- Observation - the sessions can be viewed locally or remotely;

[1] https://www.csl.illinois.edu/

- Safety - for conducting or simulating potentially hazardous experiments if conducted physically.

It may be thought that these resources will be unique to the areas of engineering or technology in what is commonly called Science, Technology, Engineering, and Mathematics (STEM) (Carnevale, Smith, & Melton, 2011). There are, however, many other areas such as education, languages, sport or medicine (Diwakar, et al., 2014) where these resources are very useful.

Also, with regard to individual or collective study and the possibilities of communication between students and between students and teachers, computers have brought radical changes. Today, it is commonplace to study in groups being physically distant, to exchange messages and notes instantly, and to share materials or perform collaborative assignments digitally. The use of social networks and digital media for group study and information exchange is also widespread, particularly among younger students (Martin, Wang, Petty, Wang, & Wilkins, 2018). The Internet is a huge source of information (although not always reliable) enabling doubts to be clarified or any information to be obtained simply and almost immediately. Despite the amount of false, biased or unscientifically validated information circulating on the Internet, there are plenty of scientific repositories and other reliable online resources, often associated with higher education institutions, official agencies or publishers, which allow reliable information to be obtained.

Despite all these possibilities and advantages, information technology also brings risks and threats to the educational process, and several barriers to the use of technology at school have been identified (Collins & Halverson, 2009). Some of the risks most frequently mentioned are the possible increase of distractions, whether with online games, surveys or interactions on social networks, the loss of manual writing practice, as well as the possibility of decreased effort and critical sense due to the tendency to look for answers online. Also digital educational resources, being richer, more animated and colourful, limit the need to resort to the imagination, and the logical-symbolic thinking inherent to computers can result in difficulty in understanding and evaluating the world (Setzer, 2001), which is counterproductive in children and adolescents. There are also other potentially harmful effects related to health issues, such as decreased physical activity, sight and posture problems.

These fears have already led to some decisions to restrict or prohibit the use, in particular of mobile devices, as recently happened in France[2]. There are other dangers associated with computer use, common to other contexts, for example, those related to safety (Schleicher, 2019), such as cyberbullying, phishing, loss of confidentiality of data, inappropriate content, or the possibility of establishing knowledge of and contact with anonymous and/or potentially dangerous people.

It is clear that in order to use and leverage these resources, physical and logistical conditions and resources are required to support the operation of the devices. In particular, the existence of networks, wired or wireless and the possibility of Internet access with adequate bandwidth and speed for use.

A booming field is that of mobile devices (Heflin, Shewmaker, & Nguyen, 2017) with numerous applications for the most varied themes and purposes, giving rise to a mode of teaching called mobile learning (m-Learning). The usage of devices that students use on a daily basis, the ease of access and utilization, are some of the factors referred to as advantages of m-Learning (Shih & Mills, 2007) and drivers of its uptake. There are empirical and exploratory studies (Sung, Chang, & Liu, 2016) that indicate that the use of mobile devices is more effective

[2] http://www.leparisien.fr/societe/interdiction-des-portables-a-l-ecole-la-loi-definitivement-adoptee-30-07-2018-7838517.php

compared with desktop or laptop computers. This effectiveness results from familiarity with the device and the ease of downloading and using apps free of charge, which allow learning foreign languages, mathematics, programming languages, attending courses online, and so on.

This reality with technology-based teaching and the possibilities of distance learning (D-Learning), has given rise to new ways of teaching and learning and to new concepts among which: e-Learning, b-Learning, m-Learning or u-Learning (Figure 2).

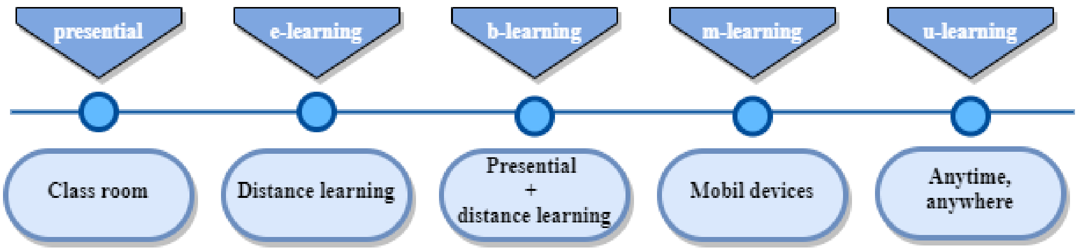


Figure 2 - Teaching Methods

It should be noted, however, that D-Learning is a very old concept, with some experiences dating back to the first half of the 19th century. With the development of the postal services, later on with radio and television, it was possible to bring education to more remote populations. In Portugal, school by TV, instituted by Decree-Law 46136[3] of 31 December 1964, was a landmark initiative, necessary to make up for the shortage of teachers resulting from the extension of compulsory schooling from four to six years. It also aimed to reduce territorial asymmetries in access to education and was crucial for rural areas (Gomes A. , 1967). Its decline began at the end of the last century with the sharp decrease in the number of students enrolled in what was then called Mediatized Basic Education (EBM). On July 8, 2003, Dispatch no. 13313/2003[4] was published, which determined the disappearance of the basic education schools that were still in operation, the last ones being closed in the 2005/2006 school year.

As far as teachers are concerned, while many of them are entrepreneurial, proactive and innovative, they train and strive to capitalise on the advantages of using computer resources, there are others who, because they cannot, dislike or even perceive technology as an obstacle rather than an ally, do not use or encourage the use of this type of resource (Muir-Herzig, 2004). There are still some cases where the facilities and resources made available do not allow them to be used.

In addition to the issues listed, technology in teaching fosters the construction of plural educational communities without geographical or cultural boundaries, which allow for the sharing of information and collaboration in a richer and more supportive context.

There are also other areas in expansion with great potential for exploration and use such as the use of podcasts, animation, video, VR, AR, mixed reality and AI.

2.1.3 The economic sector

Today, whatever the size of the business or the area of activity, it is unthinkable, even for legal reasons, not to have computer systems. For shopping, payments or doing business, you only need a device with an internet connection.

The increasing use of information technology in business has given rise to terms such as e-commerce, online shopping, Business to Business (B2B) or Business to Consumer (B2C). E-commerce is the integration between data systems, management, communication and security

[3] https://dre.pt/application/conteudo/554171
[4] https://dre.pt/application/conteudo/1234160

that enables the exchange of information on the sale of goods and services, being the electronic variant, conducted via the Internet, of any traditional business. This makes business global, regardless of the location of those involved and without distance restrictions.

This is a phenomenon that has been growing, due to its characteristics, with a reduction in costs and great scope, constituting, when well applied, an important competitive advantage (Gouveia, 2006).

B2B e-commerce refers to transactions carried out between partner organizations or in a supplier-customer rationale, typically involving raw materials or equipment, although finished products may also be marketed for resale. It is characterized by a higher volume of transactions and higher amounts regarding the operations in B2C (Turban, King, Lee, Liang, & Turban, 2015), due to the existence of several transactions of the same product along the value chain.

In the case of B2C, commercial ties are established between companies and the end customer (Nemat, 2011), usually comprising only one transaction of the same good or service. A model case of this kind of trade is Amazon. There are, however, many other examples of areas of activity such as tour operators, home banking or real estate.

In this context, consumers and businesses gain access to a global market, with the possibility of accessing a wide range of products and services, available 24 hours a day with reduced processing and distribution costs. B2C also has social benefits for society, allowing access to populations far from large urban centres or countries with less supply of goods and services.

Yet the impact of computers is not only reflected in commercial exchanges but is also an important ally for management with the provision of project management tools, work organisation, team management, document management, stock control, or data storage, among others.

Also in the industry, IT, automation, robotics and even AR (Esengün & Ince, 2018) are increasingly present and have a major impact on production capacity and product quality. The ability to control processes with the possibility of installing sensors that receive and send information in real-time allows a much more rigorous management with fewer failures and lower costs, with current industrial technologies based on sophisticated computer resources supported on IoT, Cyber Physical Systems (CPS) (Qin, Liu, & Grosvenor, 2016) (Jazdi, 2014) or Smart Connected Products (SCP) (Porter & Heppelmann, 2015). There are high-tech industries where this reality is even more evident, producing goods and equipment that are also "loaded" with technology, such as the aeronautical industry or the automobile industry.

2.2 PROGRAMMING LANGUAGES

For a computer to perform certain tasks, a program (software) is needed for the intended purpose. The software is the means by which it is possible to communicate with hardware (physical equipment) and consists of a set of instructions, which are written in a programming language.

A programming language is a written and formal language with a set of syntactic and semantic rules and its own specifications. The syntax of the language concerns the writing rules of the source code such as instructions and reserved words, whereas semantics specifies the meaning of the instructions. The definition of language is completed by a grammar that describes the syntax and semantics of that language.

The most basic way to communicate with a computer consists of a reduced set of instructions sent directly to the processor, allowing a limited and elementary number of actions to be performed. This set of instructions is called "machine code", specific to each processor and usually based on a binary code and is therefore not suitable for writing programs that translate complex algorithms. This is how an intermediate level language emerged, simpler and with greater abstraction, with binary codes being replaced by simple mnemonic instructions. These instructions are specific to each processor/architecture, and it is necessary to "translate" them into machine code using an assembler. Although it is simpler than "machine code", the assembly still has a low degree of abstraction which, on the one hand, allows access to resources at a lower level, but on the other hand it presents many limitations for the implementation of sophisticated programs.

Throughout history, languages were developed that were simpler and more powerful for the programmer, with even greater abstraction and closer to natural language. These are the so-called high-level languages, which have more features that increase the readability of code writing and minimize the probability of error. In addition, they can be independent from the computer architecture and therefore can be written on one machine and executed in others.

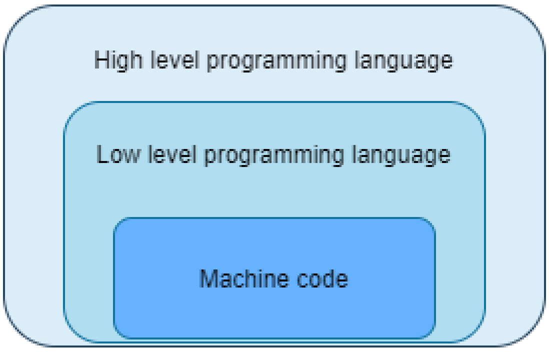


Figure 3 – Programming language levels

For the instructions to be "understood" by the processor it is necessary to convert the source code written in a specific language into machine code, and a program (or set of programs) is used for this purpose. These programs responsible for translating instructions into machine code are interpreters or compilers, depending on the language concerned.

Sebesta (Sebesta, 2016) indicates three methods of implementing programming languages (Figure 4):

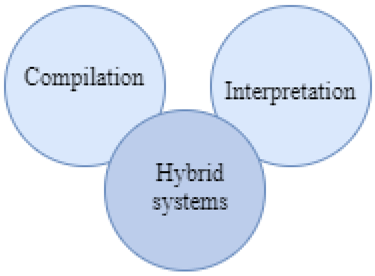


Figure 4 – Programming languages implementation methods

If the language is compiled, the source code is fully translated into a program that can be run directly (Tucker & Noonan, 2006).

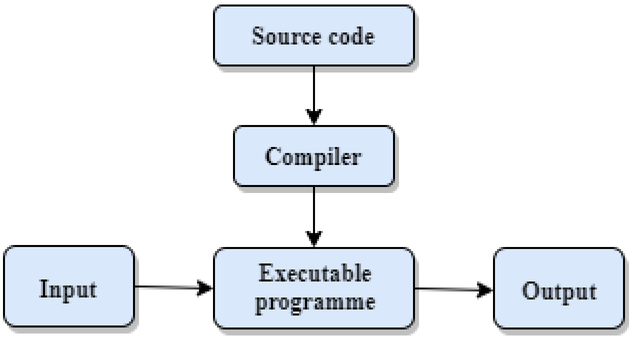


Figure 5 - Compilation process

This translation is performed to machine code, being a slow process, but because it is executed once only, the execution process is quite fast. Portability problems may arise since the machine code is different depending on the processor and computer architecture.

In the case of the interpreter, each of the instructions is translated and executed, sequentially, until the program is finished.

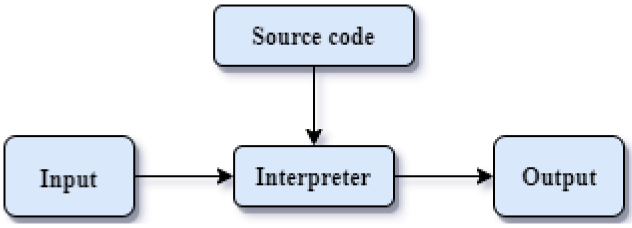


Figure 6 - Interpretation process

In these types of languages execution is slower, as it is necessary to translate the source code each time you want to run the program, typically having higher demands in terms of resource use. However, it has the advantage that the generated code is specific to the platform where it will be executed.

In hybrid systems, there is a compromise between pure compilers and interpreters, with a translation to an intermediate language that will then be interpreted in a simpler and faster way, so they are platform-independent, with average execution speed.

In Table 2 some characteristics of the methods of implementing programming languages are presented.

Table 2 - Comparison of methods of implementing programming languages

Compilation	Pure interpretation	Hybrid Implementation Systems
• Machine code translation • Slow translation • Fast execution	• Without translation • Slow execution • Falling out of use	• Low translation costs • Average execution speed

An example of a language where a process of compilation and interpretation occurs is Java[5]. In the process of switching from source code to executable code, the source code is

[5] https://www.java.com/en/

initially submitted to the Java compiler where a special representation called *bytecode* is generated as illustrated in Figure 7.

Figure 7 – Java compilation process

There is another software component called Java Virtual Machine (JVM) which consists of an application that is created on each computer when the software of the Java platform is installed there, and which is specific to the operating system of that computer. It is on JVM that the bytecode will be interpreted (or compiled) thus allowing the execution of the program written in Java (Figure 8).

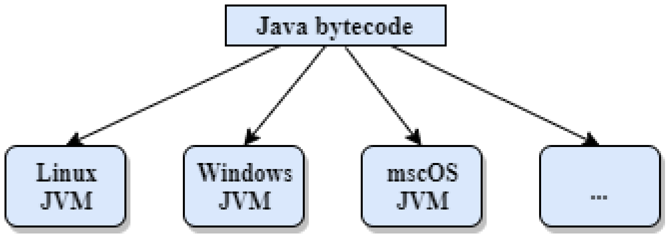


Figure 8 – Java interpretation process

Since each computer has its own JVM, this results in the advantage of code portability to any machine, regardless of its characteristics and operating system, according to the slogan "Write once, run anywhere" by the company that launched the language, Sun Microsystems which was acquired by Oracle Corporation[6] in January 2010 (Sharan, 2017).

2.2.1 Programming Languages Evaluation Criteria

In the early days of programming, applications were typically simple and very limited due to existing hardware and the development process comprising individual programmers. Currently there are teams with several programmers who develop small parts of large complex and sophisticated projects with marked quality and safety concerns. Thus, programming languages have had major developments and should today present a set of desirable properties that make them robust, safe and reliable.

For the design and implementation of a computer system, the choice of programming language is a key factor for its success. There are languages best suited for certain types of applications and environments, for Web or offline use, for handling large volumes of data, or where a great deal of calculation effort is required, among other analysis parameters. These days it is common to have applications that combine more than one programming language depending on the specificities of each functionality. Thus, the choice of language (or languages) should focus on the application, but also ease the development process.

After the so-called software crisis (Pressman, 2010), programmer time management became vitally important in the development of software, which led to programming languages allowing the optimization of this resource. However, several other characteristics were

[6] https://www.oracle.com/index.html

identified as being important, the most commonly mentioned being readability, writability (writing ability), reliability, cost, efficiency and portability (Sebesta, 2016).

2.2.2 Readability

Readability refers to the easiness of reading and understanding programs written in the language concerned, i.e. the ease of analysing the code and perceiving its logic, being a very important aspect in program maintenance. Readability comprises the following aspects: overall simplicity, orthogonality, control structures, data types and syntax.

2.2.2.1 Writing ability

This property indicates the ease of using a language to write code in a natural way, keeping the programmer focused on the content and not on the details or "tricks" of the language. It has associated factors such as simplicity, orthogonality, expressiveness, support for abstraction, data structures, control structures and syntax.

2.2.2.2 Reliability

This property is intended to assess whether, with the use of programming language, compliance with the specifications is met under all conditions (Sebesta, 2016), showing identical behaviour regardless of platform or data volume. The reliability of systems is much more dependent on algorithms than on the language with which they are written. However, the language can have some impact, being a reliable language that which promotes the implementation of applications with low error rates and maximizes the automatic detection of errors. The language must provide assurance that the programs work for the intended purpose, with data integrity and access control.

2.2.2.3 Cost

The final cost of a language is a concept that results from the sum of a set of costs, dependent on many factors such as the ease or difficulty of training programmers. There are robust but complex languages with many features and multiple ways of executing the same instructions, which makes them more difficult to learn and use. Other factors contribute to determining cost, such as ease of writing, readability, reliability, ease of execution and testing, as well as the financial costs associated with purchasing or using resources (compilers, publishers, debuggers, servers, virtual machines or other). Also, the cost associated to the maintenance of software, being very much related to readability, can be quite significant and impacting, especially in corrections and modifications made by programmers who are not the authors.

2.2.2.4 Efficiency

Efficiency in general terms concerns the relationship between the means employed and the results obtained, i.e. producing the most by 'spending' the least. In computing terms, efficiency is historically related to physical resources such as speed of execution, processor usage or memory occupied to run a program. However, the coding effort has increasingly been considered in the equation, both for the initial writing of programs and for their maintenance.

2.2.2.5 Portability

This property consists of the ability of software to run on different platforms or operating systems with no need for change, i.e. to be compatible with different environments, always displaying the same behavior. A program written on one machine should be able to be used on any other equipment.

2.2.3 Programming Paradigms

The term paradigm means, in broad terms, standard or model, i.e., it designates a conceptual rather than a concrete representation and can be described by basic guiding principles. This concept was defined in the scientific context by Thomas Kuhn, physicist and historian of science, as a set of laws, theories and methods that "provide models from which emerge specific coherent traditions of scientific research" (Kuhn, 1962), that is, a specific form of organization of knowledge in a particular area.

This concept applied in the context of computer science concerns the way in which a problem is designed and implemented, i.e. the programming style. A programming paradigm determines the programmer's view of the problem analysis, the structuring and the implementation of the solution.

The programming paradigm can also be understood as the way of organizing programming languages according to their functionalities. However, a language can be multi-paradigmatic, and the programmer can use it according to what appears to be the most appropriate for the implementation in question.

There are several programming paradigms, four of which are fundamental:

- Imperative;
- Functional;
- Logical;
- Object oriented.

Each corresponds to a specific approach involving a different way of thinking and being supported by a diversity of programming languages. This diversity impacts directly on the development process, since the choice of a paradigm establishes a well-defined path that is difficult to reconcile with other approaches. Different paradigms may privilege different aspects, such as focusing on syntax, privileging the organization of the code, or highlighting the execution model. Sometimes the distinction between paradigms is made by the programming techniques that are allowed or forbidden in each one.

2.2.3.1 The imperative paradigm (procedural)

This is the oldest paradigm (Tucker & Noonan, 2006), directly associated with the questions of hardware, having Von Neumann's machine as theoretical basis. In imperative languages the main elements are the variables and the assignment and iteration commands.

This paradigm defines a program as a sequential set of instructions that change the current state of a system (by changing the values of its variables) until a final state is reached. A detailed description of the operations necessary to solve the problem and their order of execution is carried out, the final result being the consequence of the execution of these operations.

The imperative paradigm shows high efficiency since the instructions are stored in adjacent memory cells. Also, as far as modelling is concerned, it is simple, since the structuring is similar to usual human action, with the execution of sequential tasks. Nevertheless, it has the disadvantage that relationships between inputs and outputs are not very evident, which impairs the legibility of the code, potentiating errors in its maintenance. Another aspect typically identified as negative is to focus too much on how to do it and not on what to do.

The most relevant programming languages in this paradigm are: Fortran, Algol, Cobol, Basic, C, Ada and Perl.

2.2.3.2 The functional paradigm

This paradigm advocates programs as evaluations of mathematical functions, with immutable data and no change in states, there being no commands, but only expressions. In this paradigm, any function of language is considered a pure mathematical function where, for the same arguments, it always returns the same result, defining a concrete and precise relationship between input and output and regardless of previous states.

Here the functions are used as parameters for other functions, where, by association of elementary functions, you can define more complex functions. This paradigm is closely associated with scientific and academic applications, although there are industrial applications that use it, being used in prototyping, concurrent programming and AI. The development of software is usually fast and reliable, with low error rates.

The language most commonly associated with the functional paradigm is LISP, but there are other languages such as Scheme, Haskell, Erlang or OCaml.

2.2.3.3 The logical paradigm

The logical paradigm is based on the definition of rules and restrictions without the need for detailed or ordered specification, based on the principles of formal logic. It originates in the attempt to automatically demonstrate theorems in AI (Gabbrielli & Martini, 2010).

This paradigm is based on the basic idea that new facts can be derived from existing facts and rules, supported by the logic of Frege's predicates where a predicate is the definition of a relationship between variables. Through relationships between inputs and outputs, it enables a high level of abstraction.

It is based on propositions (concrete and known facts) inference rules (where the form of deduction of propositions is established) and inference control strategies. Not deterministic, it is suitable for problems with incomplete specifications, where problems can be solved by deduction or inference.

The most representative language of this paradigm is Prolog, although there are others like Gödel, ACL2, LDL and Isabelle.

2.2.3.4 The Object-Oriented Paradigm

Although it is a widely used paradigm today, the Object Oriented (OO) paradigm already bears a long history dating back to the 1960s of the 20th century, supported by the Simula language, with the incorporation of many of the concepts still in use to this day. Later, with Smalltalk it had a new impulse, becoming widely used with the appearance of C++ in the 80s. However, its widespread and massive use came after the first OOPSLA[7] conference (Object-Oriented Programming, Systems, Languages and Applications) in 1986.

The underlying idea behind this paradigm is imported from hardware development techniques where a set of associated components constituted a more sophisticated and complex system. It aims to make the development of software faster and more reliable. The basic elements of this paradigm are the objects, which are mathematical abstractions representative of real-world elements, significant in the context of the application. An object is an entity characterized by a state (defined by the value of its attributes), by behaviours (methods) and by a unique identity. In the methods and attributes the relationship forms with other objects are also defined.

Other relevant concepts of this paradigm are, in a non-exhaustive or exclusive way, classes, hierarchy, inheritance, polymorphism, encapsulation, abstraction and modularity. A class is an

[7] http://www.oopsla.org/oopsla-history/

abstraction of a grouping of objects with common characteristics and behaviours, that is, an object project (model). An object is thus an instance of a class. The system's functioning is ensured by messages exchanged between the various objects that make it up.

The strengths of this paradigm are the great reuse of code, ease of maintenance and robustness, being the most relevant programming languages: Smalltalk, C++, Eiffel, Java, Python, C#, PHP and Ruby.

2.2.4 The importance of choosing a Programming Language

The reasons for adopting a particular programming language (or languages) can be diverse and of various kinds. A relevant and conditioning aspect is undoubtedly the end purpose. There are languages best suited to certain purposes, environments, mode of operation or operating systems, such as whether the goal is development for Web, whether you want distributed systems, whether you need to handle a large data set, whether the applications are for finance or science or whether you want to develop for mobile applications. Sometimes several needs have to be met simultaneously, which can lead to the use of several programming languages. There are also application fields such as AI, optimization, statistical treatment, system administration or computer security audit, among others, which have particularities for which it may be necessary to use specific languages.

Obviously, technical issues are also very relevant, according to evaluation criteria such as those listed above. But also, the maturity (and stability) of the language or the availability of resources such as Application Programming Interface (APIs), Software Development Kit (SDK) or even the development frameworks can have a significant weight in the decision. Also, to be considered are the issues associated with virtualization, a growing trend and a crucial turning point for the streamlining of Development and Operations (DevOps), backups and system replacement. Also, containers, kubernets, dockers, cloud computing, micro services, or complex services such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS) are an increasingly present and common reality.

There are also operational and logistical reasons to consider, for example, the existence of programmers in the organization capable of coding consistently and quickly in the language to choose, or, if not, the availability of programmers in the market. With today's extraordinary communication facilities, many organizations think of the market in a global way, considering the hiring of programmers in physically distant places. However, this practice carries some risks such as culture shocks, mismatched schedules and costs associated with communication. Moreover, if there are no programmers available on the local or regional market, this shortage is likely to be associated with an increase in cost.

In organizations where there is already a history of development, the knowledge and skills of team members, as well as the ability and ease of recruiting new programmers with knowledge of the language to be adopted, are important factors. In addition, the so-called software legacy, often thus designated with a pejorative connotation indicating obsolescence, but still in operation, sometimes has a long history of development with the participation of dozens of programmers, with numerous changes over time, but not always properly documented. Therefore, the knowledge and experience accumulated are almost exclusively in the existing software itself. While the maintenance costs of a legacy system are typically high, it is no less true that the decision to upgrade or migrate applications to new systems carries risks and will have to be well thought out in order to allow the process to be successful with technical and economic feasibility.

There are also issues related to trends or "fashions", sometimes induced by large global organisations that influence decision making.

In short, the choice of language is strongly related to the purpose of the application, its complexity and specific characteristics. However, no less important are practical and logistical issues such as human capital, development methodologies and support tools, as well as pre-existing code, programmers' experience and the existence of open source (Meyerovich & Rabkin, 2013).

2.2.5 The most relevant Programming Languages

Throughout history many hundreds of programming languages will have certainly been developed, with some claiming it to have been thousands[8] As early as 1978, Jean Sammet identified 166 programming languages in the United States alone, which were also used by more people than their authors (Sammet, 1978a). The list deliberately did not include languages from other sources because of difficulties in obtaining a complete and reliable list, which proves that there would be more to the list than was presented.

Many others will have been developed but will never have gone beyond prototypes or have been used only in restricted contexts or for specific, one-off purposes. However, others have been used over time, although several with limited usefulness and lifetime.

One question that sometimes arises is: why are there so many programming languages? Surely there will be no single answer and it will include several reasons. One is that technological evolution will explore new areas and pose new challenges, with new problems to be solved, leading to the development of new tools. Also, the fact that the first languages have existed for dozens of years makes them obsolete and unsuitable for new realities of *hardware* as well as for methods and techniques of problem solving in some contexts. It may also result from an option whereby it is understood that existing languages are not sufficiently suitable for the intended purpose or because research projects are intended to open up new horizons and have to be sufficiently innovative to attract funding (Sammet, 1972).

Some are more important, others less suitable for specific purposes or general use, compiled, interpreted or hybrid, for proprietary systems or for open systems, there are programming languages for the most varied purposes and contexts (Parker & Davey, 2012) (Kumar & Dahiya, 2017).

In the following figure, a summary is presented that highlights some of the main high-level programming languages from 1957 to the present (Lovrenčić, Konecki, & Orehovački, 2009).

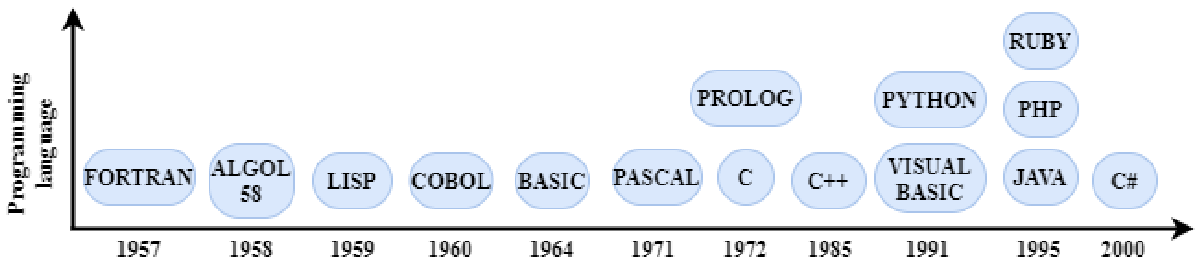


Figure 9 – Evolution of Programming Languages

A brief description of each of these languages will follow, in chronological order, their history, their characteristics and their contribution to the writing of computer programs and to the development of information technology.

[8] http://www.info.univ-angers.fr/~gh/hilapr/langlist/langlist.htm

2.2.5.1 FORTRAN

The name FORTRAN results from the acronym obtained from the name of the IBM project Mathematical ***FOR***mula **TRAN**slating System, developed at IBM in the 1950s. This project, led by John Backus[9] led, in 1957, to the FORTRAN language (Adams, et al., 2009). It is considered the first high-level language and was developed for scientific applications related to mathematics where a large volume of calculations on a limited set of data were intended (Backus, 1978). Although it had many limitations, notably in handling strings and in data structures, it represented a major technological leap forward in that it made the programmer's function much easier, allowing them to concentrate more on solving the problem than on the coding task.

As it was developed specifically for IBM, other companies which started using FORTRAN developed their own compilers, which made portability impossible. To address this problem, the American Standards Body, predecessor of the American National Standards Institute (ANSI), promoted the publishing of a standardised version called FORTRAN IV. Currently, FORTRAN is still used having had several versions over time, one of the most famous being FORTRAN77 and the most recent FORTRAN 2018[10].

2.2.5.2 ALGOL

ALGOrithmic Language (ALGOL) is a programming language developed by European and American scientists, a FORTRAN descendant, and whose first version became known as ALGOL 58 (Backus, et al., 1963). It was presented at a conference in Zurich (Backus, 1959), and was initially named IAL (International Algebraic Language). However, this acronym was considered pompous and difficult to pronounce (Perlis, 1995). John Backus (O’Regan, 2013), an American scientist co-author of FORTRAN, and one of eight scientists involved in ALGOL development, is the author of a method used for the description of ALGOL 58, which, together with Peter Naur would result in the so-called Backus Naur Form (BNF). This is a grammar syntactic description technique, widely used to express the syntax of a programming language.

The subsequent version, ALGOL 60 (Backus, et al., 1963), would become the standard of ALGOL and its use became widespread, having had great use in scientific applications, with greater incidence in the field of mathematics. Its use in the development of applications outside the context of science was very rare, but it was of great theoretical and historical importance for the description of algorithms up to about 20 years ago. It has also had a great impact on the design of other programming languages, in particular the emergence and development of imperative paradigm languages (O’Hearn & Tennent, 1997).

The latest version (ALGOL 68) was not consensual and gave rise to much criticism and would be used very little (Lindsey, 1996).

2.2.5.3 LISP

LISt Processing (LISP) is a programming language created in 1959 by John McCarthy[11], a professor at Massachusetts Institute of Technology (MIT) (Lovrenčić, Konecki, & Orehovački, 2009), the first language used in AI. It originated from a FORTRAN extension called Fortran List Processing Language (FLPL). It was, at the time, an innovative language in several aspects, introducing many concepts still in force today, such as, for example, garbage collection, developed to solve memory management problems. Sometimes it is designated not

[9] https://www.computerhistory.org/fellowawards/hall/john-backus/
[10] https://wg5-fortran.org/f2018.html
[11] https://www.computerhistory.org/fellowawards/hall/john-mccarthy/

as a language, but as a family of languages, as it has given rise to several dialects such as MacLisp, Common Lisp, Scheme, Interlisp and Clojure, among others. It is considered the first functional language and will have given rise to all the others of the functional paradigm.

LISP is a language suitable for symbolic processing, being purely recursive and non-iterative (Sebesta, 2016). It is one of the oldest programming languages with some relevant use and is probably the most widely used in the AI world.

2.2.5.4 COBOL

This language was designed to support the development of commercial applications, COBOL being the acronym for COmmon Business Oriented Language. It was developed in the context of a consortium formed in 1959 called Conference on Data Systems Languages (CODASYL), sponsored by the U.S. Department of Defense and which included both scholars and industry. Although not directly linked to the project, Admiral Grace Murray Hopper is considered to have been "the grandmother of COBOL" (Gürer, 1995) for her previous contributions, including the development of FLOW-MATIC, the first language to use instructions with English words, and the influence she had on the development of COBOL. CODASYL was formed with the intention of developing a programming language, not proprietary, that could be used in any computer, serving as a universal means of communication, in what represented one of the first examples of attempts at standardization (Sellink & Verhoef, 1997). The assumptions for its development included the processing of large volumes of data, but with few computational requirements, and was therefore not suitable for scientific use or for complex calculations. It should be based on the English language and should be easy to read, having been structured like a written report. This has made its writing more difficult by strongly conditioning its writability. The language has a hierarchical structure that contemplates several levels, in which the first are divisions, sections and paragraphs, having four main divisions: identification, environment, data and procedures (Sammet, 1978b).

The consortium's work resulted in the first COBOL compiler in 1960, marking a historical fact of portability, when the same program written in COBOL was run on computers from different manufacturers, by only changing the environment division. The first standard was COBOL 60, quickly replaced by COBOL 61, and over time by several others such as COBOL 74, COBOL 85 and COBOL 2002 (Manev & Maneva, 2014). It also gave origin to several dialects of different companies, with several versions and updates that were improving its operation, increasing its potential and integrating new features such as Object Orientation.

2.2.5.5 BASIC

This is one of the most famous languages in the history of computers, popularized in the 80's of the 20th century, popularized by its dialect Sinclair BASIC, used in ZX Spectrum. BASIC stands for Beginner's All-purpose Symbolic Instruction Code, i.e. a general language for beginners.

BASIC was developed in the 1960s of the 21st century, at the American University of Dartmouth[12], by professors John Kemeny and Thomas Kurtz and with the collaboration of a group of students (Kurtz, 1978). Another relevant name in this context is that of Mary Kenneth Keller, a Catholic nun pointed out as probably being the first American woman to obtain a doctorate in computing (Gürer, 1995).

The decision to create a new language resulted from the finding that existing languages were too complicated, only understandable by mathematicians and scientists, and from the need

[12] https://home.dartmouth.edu/

to open up programming to other profiles as students of the human sciences or the arts. Some requirements were defined for the development of BASIC, among which: it must be general-purpose, easy to learn and use, interactive, used both for educational purposes and also for scientific purposes, allowing advanced functionalities, but without making it too complicated and independent from hardware.

In its genesis it has a revolutionary idea of resource sharing, with an architecture in which several terminals shared access to the same computer, with the time of use being distributed among the terminals. The 1st of May 1964 is considered an historic day for the University of Dartmouth and for BASIC, because it was at 4:00 a.m. on that day that the first program written in BASIC (Kurtz, 1978) was executed simultaneously in two terminals.

In 1966, the authors allowed a universal use of the language, and it became very popular in the 1970s and 1980s of the 20th century. It began to lose influence in the early 90's of the same century, with the increase in capacity of hardware and with the advent of other languages, in particular its direct successor, Visual BASIC, which already incorporated visual programming resources and OO.

2.2.5.6 PASCAL

The name of this language is a tribute to the French physicist and mathematician of the 17th century, Blaise Pascal, who invented the first calculator. Pascal was developed by Niklaus Wirth[13], a Swiss professor at the Federal Institute of Technology in Zurich in the early 1970s of the 20th century, with the aim of being used in the teaching of structured and modular programming (Wirth, 1971). Between 1977 and 1985 several books on Pascal were published, in several languages, promoting the language and making it known and used worldwide, being also, in the mid-1990s of the 20th century, the most used programming language in teaching (Wirth, 1996). This is proven by a survey conducted in 1996 which concluded that Pascal was used in about 36% of educational institutions, the second language being C++ with 32% (Parker & Davey, 2012).

Pascal is a language based on ALGOL, but with great concerns for simplicity, with a small number of instructions and with great emphasis on syntax, which has made it somewhat inflexible. It was one of the first languages with structured programming concerns, having pioneered several aspects such as the dynamic allocation of memory by using pointers, reading and writing procedures of individual fields or the use of recursiveness (Lovrenčić, Konecki, & Orehovački, 2009). Although its original field of use is education alone, it has gained notoriety, especially after it began to be used at the University of California, becoming a language of general use, being at the genesis of other languages such as Ada and Modula-2 (Lovrenčić, Konecki, & Orehovački, 2009).

2.2.5.7 C language

C is a widely used language (Norrish, 1998) and is probably one of the best known and most important programming languages in history (Krause, Larisch, & Salfelder, 2019). It originated at AT&T Bell Labs (American Telephone and Telegraph) in the early 1970s and was designed for the development of operating systems, being the basis of the UNIX operating system (Ritchie, 1993). Despite its origin and usefulness for operating systems and compilers, it has evolved into a general-purpose language, being used in several domains and types of applications.

[13] https://www.computerhistory.org/fellowawards/hall/niklaus-wirth/

Created by Dennis Ritchie[14], the C language was derived from the B language (Johnson & Kernighan, 1973), also by Ritchie and Ken Thompson, which in turn was a simplification of Basic Combined Programming Language (BCPL*)* by Martin Richards, University of Cambridge (Richards & Whitbey-Strevens, 1979). Unlike its predecessors, C language is a "typed" language, although not strongly "typed"; however, throughout its evolution the verification of types (Kernighan & Ritchie, 1988) has been reinforced, introducing new types of primitive data, such as whole numbers, characters, matrices, pointers and special Boolean operators.

The C language was defined by ANSI X3.159-1989 (ANSI, 1989) and later received certification from the International Organization for Standardization, ISO/IEC 9899:1990[15], the most current version being that of 2018[16]. This standardization decision was made to promote the C language, giving it reliability and capacity for maintenance and execution in different systems.

The objective of its creation was to be "minimalist", simple and flexible, with a restricted set of instructions. It has a significant set of resources, giving the programmer great power, having as its basic philosophy the programmers' responsibility.

C language is easy to learn and has been widely used for teaching programming and has proven to be a fast, reliable and effective language over time, usable for a wide range of purposes (Papaspyrou, 1998).

2.2.5.8 PROLOG

Prolog, by Alain Colmerauer and his PhD student Philippe Roussel, resulted from a project whose first objective was not to develop a new programming language (Colmerauer & Roussel, 1996). Its purpose was to develop a software tool to implement a human-machine communication system in natural language, in this case in French, based on the idea of automated deductive reasoning. From this project, developed at the University of Marseille, the first version of what would be called Prolog (PROgrammation en LOGique) resulted, named after Roussel who worked with Jean Trudel from the University of Montreal on the deduction part of the project. Colmerauer, in collaboration with Robert Pasero, was responsible for the natural language part. Later Robert Kowalski, from the University of Edinburgh (Colmerauer & Roussel, 1996), joined the project, considered one of the founders of logical programming, author of resolution SL (Kowalski & Kuehner, 1971), and who was invited by the importance of his work of semantic formalization of programming with Horn clauses. Prolog solves problems by using techniques originally developed to prove theorems in logic.

A historical landmark in the history of Prolog was the specification of the first standard, the so-called Edinburgh Prolog, by David Warren and the Portuguese Fernando Pereira and Luis Moniz Pereira (Warren, Pereira, & Pereira , 1977).

Prolog has been widely used in the field of theorem proofing, natural language processing, knowledge-based agents and in the field of AI in general, also influencing the development of new programming languages, Erlang.

2.2.5.9 C++

C++ was conceived as an extension of the C language and, like its predecessor, was developed at AT&T's Bell Laboratories. The project, started in 1979 by Danish scientist Bjarne

[14] https://www.computerhistory.org/fellowawards/hall/dennis-ritchie/
[15] https://www.iso.org/standard/17782.html
[16] https://www.iso.org/standard/74528.html

Stroustrup[17], was first named "C with classes" (Stroustrup, 1993). Stroustrup's motivation for the creation of a new language originated in his PhD work at Cambridge University, where he analysed Simula language as well as C language and its predecessors (B and BCPL) (Richards & Whitbey-Strevens, 1979). C++ results from the attempt to use the best features of C and Simula, avoiding its inconveniences.

In 1983 "C with classes" was renamed C++, as an allegory to the evolution of C, and the first commercial version was made available in 1985, when the first edition of the book The C++ Programming Language (Stroustrup, 2013) was also published. The author states that "The main goal of C++ is the writing of good programs, in an easier and more pleasant way for the individual programmer" (Stroustrup, 1993).

C++ is a general-purpose language that presents characteristics of the imperative programming paradigm, but also of the OO paradigm, which has become quite popular, mainly for its compatibility with the C language and the existence and availability of good free compilers.

2.2.5.10 VISUAL BASIC

Although Visual Basic appeared in 1991, its history began a few years earlier, in 1987, when Alan Cooper[18] developed Ruby, a visual programming software (Waite, 1992). After a demonstration for Microsoft, Bill Gates considered it a significant innovation, buying it and incorporating it into the company. This would be an important step towards the emergence of Visual Basic as an evolution of BASIC, it being specially designed for Microsoft Windows.

The creation of Visual Basic was a milestone in the history of computer science with the emergence of a language with innovative features because it is event-driven (event-driven) and with resources for graphical interfaces. It has an integrated development environment (Integrated Development Environment - IDE), providing the interface in which the programmer creates his application. Given the growing popularity of graphical interfaces at the time, it was also an important product for Microsoft.

The language had several evolutions over time, with version 6 (VB 6.0) being released in 1998. From then on it was replaced by Visual Basic.NET (Shapiro, 2002), being currently part of this framework.

2.2.5.11 PYTHON

The Python[19] language by the Dutchman Guido van Rossum[20] began to be developed in 1989 and was presented in 1991. To name the language the author was looking for a short and mysterious name, having been inspired by Monty Python's Flying Circus, a comedy programme that was then broadcast by the BBC.

Python was designed with simplicity, readability and clarity in mind, and it should be "minimalist", allowing programs to be written with fewer lines of code than existing languages (Summerfield, 2007).

In the development of Python, its author established the following objectives:

- To be a simple and intuitive but powerful language that could stand shoulder to shoulder with its main competitors;
- To be open source, enhancing its development through diverse contributions;
- Be easily understood, with good readability;

[17] https://www.computerhistory.org/fellowawards/hall/bjarne-stroustrup/
[18] https://www.computerhistory.org/fellowawards/hall/alan-cooper/
[19] https://www.python.org/psf/
[20] https://www.computerhistory.org/fellowawards/hall/guido-van-rossum/

- To be of general use;
- To allow writing code easily and quickly.

Python's syntax derives from that of other languages having strong influences from ABC, C language, Modula-3 and Icon (van Rossum, 1995).

Over the years Python has had several evolutions, today it is a flexible language, with simple syntax, easy to learn and which allows a fast development, as intended by van Rossum. Although in its genesis is the OO paradigm, it supports other programming paradigms as imperative and functional (Rashed, 2012).

In 2008 Python 3.0 was released and it is considered the future of Python, the most current version being 3.7 (June 2018).

Python is now considered more than a programming language, consisting of an ecosystem with numerous sophisticated libraries and specialized commands developed specifically for dozens of different areas. It supports applications from several companies and in a wide range of applications, and is also widely used by universities and other educational institutions for research and teaching programming (Lutz, 2013).

2.2.5.12 JAVA

Although Java formally appeared in 1995, its origins date back to the early 1990s when James Gosling[21] of Sun Microsystems[22] began developing Oak to add functionality to C++. The idea was to develop a new technology for networking for embedded systems and electronic devices (Horstmann, 2013). Initially, the use of C++ was considered, but the idea was abandoned due to technical difficulties for the intended purpose. Thus, an object oriented language was born, with similarities to C++, but simpler, safer, more portable and reliable (Arnold, Gosling, & Holmes, 2015).

Java source code is stored in files with .java extension, which are compiled in bytecode format (files with .class extension) and can be executed by an interpreter. Java has great portability, because each operating system has its own JVM where the bytecode will be executed, although it can also be converted directly into machine language by a just-in-time (JIT) compiler, increasing efficiency by decreasing execution time.

In order to use Java, it is necessary to have a JDK (Java Development Kit) which, besides a compiler, provides an extensive library of classes which ensure simple tasks during the development of the application.

The use of Java for application development presents as strengths (Deitel & Deitel, 2011):

- Simplicity, though also robust;
- Object-oriented;
- High level of abstraction;
- Use of references;
- Absence of pointers, limiting the programmer's action and increasing safety;
- Portability;
- Concurrent processes (threads);
- Garbage collection*;*
- Syntax similar to some older languages, making it easier for programmers to migrate existing code and adapt;

[21] https://www.computerhistory.org/fellowawards/hall/James-Gosling/
[22] https://www.oracle.com/sun/

- Existence of free tools for compilation and debugging.

For these reasons Java presents itself as a flexible and very popular language, in particular due to the advent of the internet and the development of applications for mobile devices, being used in several types of applications and by programmers from all over the world. Currently Java is maintained by Oracle[23], after the purchase of Sun Microsystems (Sharan, 2017).

2.2.5.13 PHP

PHP was introduced in 1995, its author being the Canadian naturalized Danish, Rasmus Lerdorf. PHP originally meant Personal Home Page, and was renamed as a recursive acronym for Hypertext Preprocessor (Hudson, 2005). It was created to facilitate several programming tasks for Web, in order to make them less repetitive, reducing the size of the code to write (Sklar & Trachtenberg, 2014). It should be simple by providing various features for various purposes (Lovrenčić, Konecki, & Orehovački, 2009).

It is an open source scripting language for the production of dynamic pages, working embedded in Hyper Text Markup Language documents (HTML) and hosted on the server side, which interprets the code and produces an output, on the client side, depending on the data currently used (Lerdorf, 2000).

Throughout history, PHP has had several versions and has undergone major evolutions. Due to its great popularity, in 1997 it ceased to be a personal project of Lerdorf becoming a very relevant technology for the Internet incorporating contributions from various programmers (Castagnetto, Rawat, Schumann, Scollo, & Veliath, 1999). In 1998 the PHP3 version is released, rewritten by Andi Gutmans and Zeev Suraski (Lovrenčić, Konecki, & Orehovački, 2009), refining the syntax and incorporating several improvements, giving it a configuration very close to what still exists today.

Another important milestone in the history of PHP was the creation of an organization that would be called PHP Framework Interoperability Group (PHP-FIG)[24] in 2009. PHP-FIG would create PHP Standards Recommendations (PSRs), a standard element of concepts that aims to create a common technical basis for writing code in PHP.

PHP is widely used in Internet sites, being present in almost 4/5 of all sites[25]. The most current version of PHP is PHP7, released in December 2015, although the most widely used version is still PHP5.

2.2.5.14 Ruby

This language was thus named by its author, the Japanese Yukihiro "Matz" Matsumoto, as a result of a joke with some of his friends (Lovrenčić, Konecki, & Orehovački, 2009). Even before writing any code, Matsumoto thought about creating a new programming language whose name should be that of a precious stone, designating it Ruby in honour of a friend born in July, since the ruby is the stone of that month[26].

The development began in 1993 and the language was publicly presented on December 21, 1995[27]. Matsumoto intended a scripting language, object oriented, having identified Python and Perl as potentially interesting, but none of them satisfied him completely. He felt that, in general, the existing languages were hard, ugly and too limited or too complex. For this reason, he conceived a language with the objective of being more object oriented than Python and more

---

[23] https://www.oracle.com/index.html
[24] https://www.php-fig.org/
[25] https://w3techs.com/technologies/details/pl-php/5/all
[26] https://www.americangemsociety.org/page/birthstones
[27] https://www.ruby-lang.org/en/about/

powerful than Pearl. Another of his goals was that the programming should be easy and, moreover, fun, having as purpose in Ruby's design that the language "would make the programmers happy" (Flanagan & Matsumoto, 2008).

This is an interpreted, object-oriented, multi-platform, open-source, general-purpose programming language with a wide range of applications. As in other script languages like Python or Perl, in Ruby, coding is done with a small number of instructions, yet allowing good readability of the code.

ISO/IEC 30170:2012[28] specifies the syntax and semantics of Ruby, as well as the compliance requirements of processors and the compliance of programs written in Ruby. There are several implementations of Ruby, many of which are open source. However, the reference implementation is the so-called "Matz Ruby Implementation (MRI)" (ISO-International Organization for Standardization, 2012) and all others should check its compatibility against it.

2.2.5.15 C#

The origin of C# dates back to 1999, when Microsoft created a team to create a new programming language. The team, led by Danish Anders Hejlsberg, author of Turbo Pascal and director of the Delphi project, developed a language based mainly on C, C++ and Java[29] which they called C-like Object Oriented Language (COOL) (Buono, 2005). In July of the following year, at Microsoft's Professional Developers Conference, the language was to be presented, already with the designation of C#, and at the same conference, the framework .Net of Microsoft was also introduced, for which C# was developed. The need to change the name resulted from COOL issues already existing as a trademark. Thus, C# was adopted, supposedly in an allusion to music, where the sign # means sharp, which represents half a tone above the preceding note. Thus, C# being the successor to C++, would be above it. There are, however, those who claim that the explanation, being similar, is different, resulting in C++ as an increment (of one unit) and C# as an increment of C++, in that # can be broken down into four "+" signs.

In 2001, Microsoft submitted a request to European Computer Manufacturers Association (ECMA) for a C# standardization that would be granted by the ECMA-334[30] specification, which later became the ISO/IEC 23270:2006 (Jackson S. , 2016) standard.

In the C# development project, objectives were set, among which, that the language should be general-purpose, simple, and object-oriented. It also favoured the portability of the code, as well as the ease of adaptation of users familiar with C and C++. C# is a strongly "typed", multi-paradigmatic language, with verification of usage of uninitialized variables and with garbage collector and is suitable for the development of components for distributed environments.

2.2.6 The Programming Languages most used nowadays

Apart from the languages mentioned, there are many others, the overwhelming majority being more recent and with a use that is beginning to be relevant. These can include, for example, Ada, Delphi, Go, Groovy, Haskell, Julia, Kotlin, Lua, MATLAB, Objetive-C, PERL, R, Ruby, Scala, SQL or Swift (Lovrenčić, Konecki, & Orehovački, 2009). Some of them are of general use, but others are intended for specific purposes, some even for niches, so it is not easy to make a direct comparison as regards their use and popularity.

The popularity of programming languages is a controversial subject that can be analysed from several points of view. There are several indexes and rankings of popularity of programming languages, with different periodicity and analysis criteria, so it is not surprising

[28] https://www.iso.org/standard/59579.html
[29] https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction
[30] http://www.ecma-international.org/publications/standards/Ecma-334.htm

that the results between them are not the same. Nevertheless, it is possible to observe trends and identify, in general terms, the most popular programming languages today.

Among the most relevant sources of data on the use of programming languages for the development of software at the international level the following can be mentioned: IEEE Spectrum[31], GitHut (from GitHub)[32] [33], PYPL[34], RedMonk[35] and TIOBE[36] (Kumar & Dahiya, 2017) (Singh, 2017) (Rabai, Cohen, & Mili, 2015).

2.2.7 The IEEE Spectrum index

The IEEE Spectrum is a ranking established by the Institute of Electrical and Electronics Engineering (IEEE)[37] which is, according to them, "the world's largest technical and professional organization dedicated to advancing technology for the benefit of humanity". This ranking analyses more than 300 programming languages from nine different sources using 11 criteria (Diakopoulos, Cass, & Romero, 2014). Among the sources consulted are GitHub, Google, Google Trends, Stack Overflow[38] as well as social networks, in particular Twitter. Data from employment sites, in particular CarrerBuilder[39], where analysis of job offers to programmers and the most requested languages are performed, are also considered. Also, with regard to the academic and scientific environment, trends are analysed based on the IEEE Xplore Digital Library[40] accessing data from millions of publications from numerous conferences and scientific publications, particularly in the field of Engineering.

This index presents several "views", the main one being the global aggregate, but allowing to visualize indexes by trends (growth rates), job offers or other parameterizable by the user. It also presents, associated to each language, the purpose(s) for which it is intended, namely: Internet, mobile devices, enterprises (general and scientific use) and embedded (device control). The information is updated on an annual basis. As an example, in Figure 10 the top ten languages of the ranking IEEE are presented.

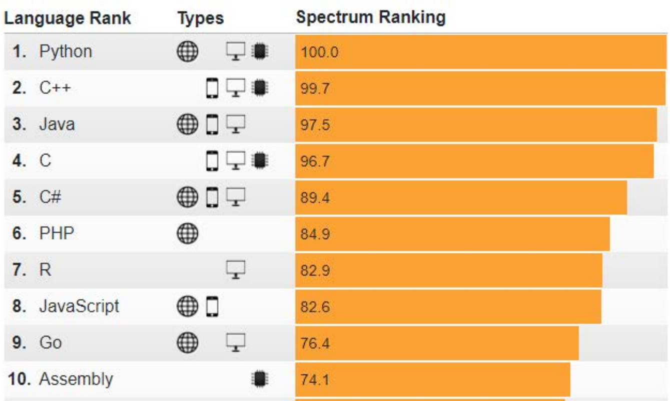


Figure 10 – Ranking IEEE Spectrum – 2018[41]

[31] https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018
[32] https://githut.info/
[33] https://madnight.github.io/githut/#/pull_requests/2019/2
[34] http://pypl.github.io/PYPL.html
[35] https://redmonk.com/
[36] https://www.tiobe.com/tiobe-index//
[37] https://www.ieee.org/
[38] https://pt.stackoverflow.com/
[39] https://www.careerbuilder.com/
[40] https://ieeexplore.ieee.org/Xplore/home.jsp
[41] https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

2.2.7.1 The GitHut 2.0 index

GitHub[42] is an online source code hosting platform, with millions of users worldwide, where it is possible to store copies of local repositories, enabling the collaborative development of software (Bell & Beer, 2015). It is based on Git, a distributed version control system, allowing monitoring the evolution of projects over time, as well as their entire history (Kelleher, 2014). Furthermore, it allows to receive contributions for open source projects, enabling code sharing and project dissemination.

Based on the repositories hosted at GitHub and the various programming languages used in them, GitHut was created in an attempt to exploit this data and obtain information about the use of programming languages. The GitHut project has had no updates since 2014, so in an attempt to continue it, GitHut 2.0 was born. This project analyses which languages are used at GitHub to determine their popularity, and the statistics are updated on a quarterly basis. In Figure 11 the most recent ranking of GitHub can be seen.

# Ranking	Programming Language	Percentage (Change)	Trend
1	JavaScript	19.922% (-2.425%)	
2	Python	17.803% (+1.519%)	
3	Java	10.482% (+0.591%)	
4	Go	7.916% (+0.295%)	
5	C++	7.253% (+0.167%)	
6	Ruby	6.296% (-0.249%)	
7	PHP	5.515% (-0.285%)	
8	TypeScript	5.415% (+0.641%)	
9	C#	4.001% (+0.659%)	
10	C	3.190% (+0.248%)	

Figure 11 – Ranking GitHut – second quarter of 2019[43]

2.2.7.2 The PYPL index

The PopularitY of Programming Language Index (PYPL index) determines the ranking of popularity of languages based on the analysis made by the frequency of search in Google of the respective tutorials, being the data obtained through Google Trends. The search is carried out on the basis of some criteria, such as using not only name, since there are names of languages such as BASIC, Java, Julia, Python or Rust, which in other contexts may have other meanings. Currently, the index has monthly updates and includes 23 languages.

According to this source, the language with the highest growth in use in the last five years is Python, with Java losing popularity, as can be seen in Figure 12.

[42] https://github.com/
[43] https://madnight.github.io/githut/#/pull_requests/2019/2

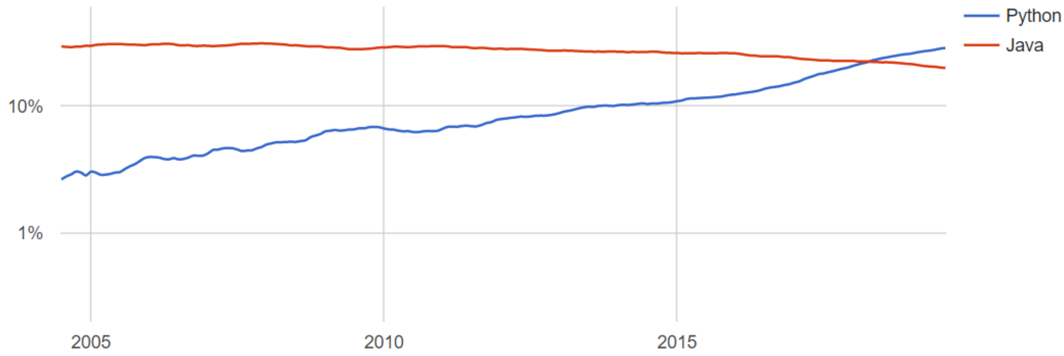


Figure 12 - Variation of use of the most popular languages

In Figure 13, the top ten positions of this ranking are presented, as well as the variation of the share of each in relation to the same month of the previous year.

Worldwide, Aug 2019 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.73 %	+4.5 %
2		Java	20.0 %	-2.1 %
3		Javascript	8.35 %	-0.1 %
4		C#	7.43 %	-0.5 %
5		PHP	6.83 %	-1.0 %
6		C/C++	5.87 %	-0.3 %
7		R	3.92 %	-0.2 %
8		Objective-C	2.7 %	-0.6 %
9		Swift	2.41 %	-0.3 %
10		Matlab	1.87 %	-0.3 %

Figure 13 - Ranking PYPL - August 2019[44]

2.2.7.3 The RedMonk index

RedMonk is a Portland-based North American software market analysis company. Its analyses are essentially based on software development projects and the activity of programmers at GitHub and Stack Overflow. The choice of GitHub is justified by its size and the volume of transactions it carries out. As for Stack Overflow it is used because it is a very active discussion forum with a large number of participants. By correlating the use of languages with the forum discussion, statistics are obtained and trends in the use of programming languages are inferred.

RedMonk's rankings consider 20 programming languages, are updated every six months, and are established based on information such as: the geographical origin of programmers,

[44] http://pypl.github.io/PYPL.html

computer technicians, system administrators, designers or database administrators, their interests or what they read and research.

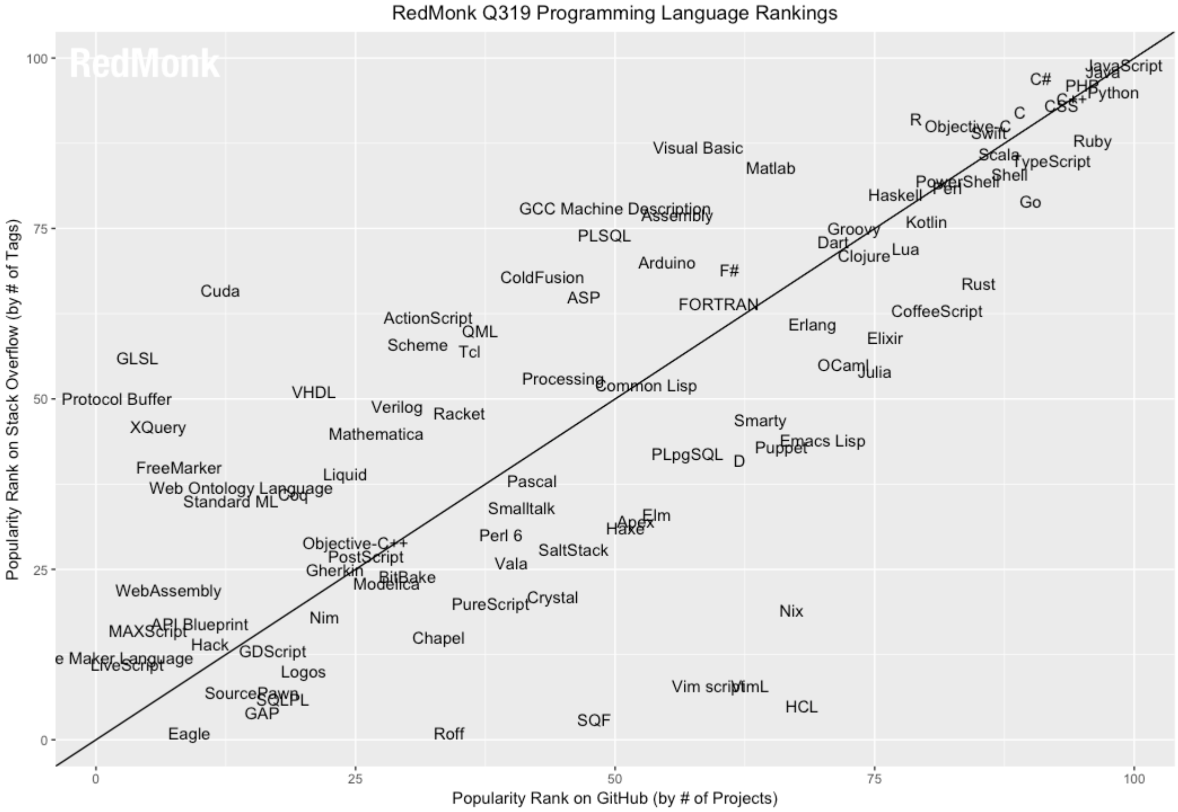


Figure 14 – Ranking RedMonk – first half of 2019[45]

2.2.7.4 The TIOBE index

One of the oldest and most considered indexes is TIOBE, by the Dutch company of the same name, specialized in software quality[46], which analyses, daily and in real time, more than 300 million lines of code. The TIOBE index is updated monthly based on the number of online surveys conducted on a given programming language. This analysis is performed on 25 sites with search engines that use a fixed algorithm the most important being: Google.com, Baidu.com, Wikipedia.org, Yahoo.com, Csdn.net and Bing.com (Kumar & Dahiya, 2017). For the language to be considered in the ranking some requirements must be met, namely the existence of a Wikipedia entry, having at least 5 000 hits on Google and being Turing complete. Due to its already relatively long existence and accumulated track record, TIOBE is considered a reliable index, enjoying a good reputation.

[45] https://redmonk.com/sogrady/2019/07/18/language-rankings-6-19/?utm_source=rss&utm_medium=rss&utm_campaign=language-rankings-6-19
[46] https://www.tiobe.com/company/about/

Aug 2019	Aug 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.028%	-0.85%
2	2		C	15.154%	+0.19%
3	4	^	Python	10.020%	+3.03%
4	3	v	C++	6.057%	-1.41%
5	6	^	C#	3.842%	+0.30%
6	5	v	Visual Basic .NET	3.695%	-1.07%
7	8	^	JavaScript	2.258%	-0.15%
8	7	v	PHP	2.075%	-0.85%
9	14	^^	Objective-C	1.690%	+0.33%
10	9	v	SQL	1.625%	-0.69%

Figure 15 - Ranking TIOBE - August 2019[47]

As already mentioned, and as can be seen from the analysis of the various indices, the popularity of a language is neither consensual nor easily comparable, due to the diversity of sources, analysis methodologies, periodicity of rankings and, in some cases, grouping of similar or derived languages. In addition, the languages most commonly used are not necessarily those most in demand for the recruitment of programmers, nor those used for teaching programming. It is therefore not possible to state which programming language is most used, but it is possible to identify a number of the most relevant and popular.

For this analysis, the most recent data from the above indexes were used, considering only the 12 most commonly used programming languages, which appeared in all the indexes.

To determine the position of each language, the average position was calculated from the information of its position in each ranking, thus obtaining the results in Table 3.

Table 3 - Popularity of Programming Languages

Language	IEEE Spectrum	GitHut 2.0	PYPL	RedMonk	TIOBE	Average position
Python	1	2	1	3	3	2,0
Java	3	3	2	2	1	2,2
JavaScript	8	1	3	1	7	4,0
C++	2	5	6	5	4	4,4
C#	5	9	4	6	5	5,8
PHP	6	7	5	4	8	6,0
C	4	10	6	9	2	6,2
Ruby	13	6	12	8	11	10,0
Go	9	4	15	16	17	12,2
Swift	18	13	9	11	18	13,8
R	7	29	7	15	20	15,6
MATLAB	11	35	10	23	12	18,2

[47] https://www.tiobe.com/tiobe-index/

As can be seen, clearly the most popular languages are Python and Java, appearing between first and third place in all rankings. As an example of the historical evolution, the register of the main languages in the TIOBE index since 2002 is presented in Chart 1, with the supremacy of Java and the C language being observed, although with a decreasing tendency, with the rise of Python being observed in recent years.

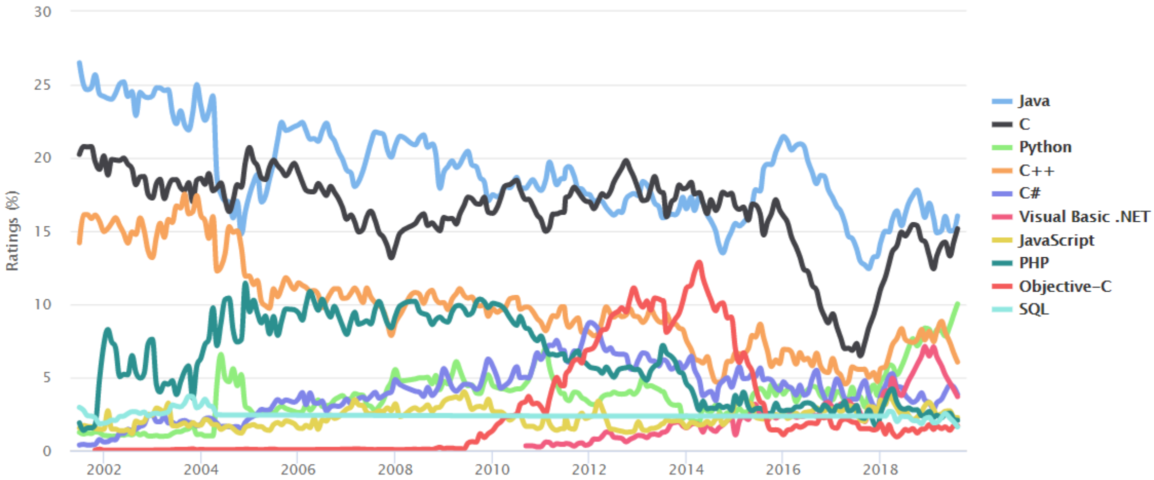


Chart 1 - TIOBE History of Programming Languages Popularity[48]

Figure 16 shows TIOBE's history of five-year popularity of programming languages since 1989.

Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2019	2014	2009	2004	1999	1994	1989
Java	1	2	1	1	14	-	-
C	2	1	2	2	1	1	1
Python	3	7	5	7	24	21	-
C++	4	4	3	3	2	2	2
Visual Basic .NET	5	9	-	-	-	-	-
C#	6	5	6	6	19	-	-
JavaScript	7	8	8	8	16	-	-
PHP	8	6	4	5	-	-	-
SQL	9	-	-	89	-	-	-
Objective-C	10	3	31	38	-	-	-
Perl	16	11	7	4	3	10	22
Lisp	32	13	19	13	12	5	3
Pascal	220	16	14	88	6	3	20

Figure 16 - TIOBE History of Programming Languages Popularity

[48] https://www.tiobe.com/tiobe-index/

These rankings, together with other criteria, can help in choosing the language to start a new software development project or in deciding to study a particular programming language.

2.2.8 Programming Languages in Education

In the context of teaching programming, there have been, throughout history, various trends using various languages. There are also courses and course units with specific characteristics and needs that condition the choice of programming language or languages. There are also other factors that contribute to the choice of the first programming language, from logistical conditions, availability of resources, knowledge and preferences of teachers (Pears, et al., 2007), to employability, trying to meet the needs of the market (Parker & Davey, 2012). Issues related to the complexity of language and the impact it may have on students' motivation are also usually taken into consideration (Luxton-Reilly, et al., 2018), as well as possible articulation with subsequent course units. The issue of language choice is an old one (Siegfried, Greco, Miceli, & Siegfried, 2012) and the target of several papers, many of which compare languages with each other and, in most cases, do not find important differences in learning outcomes, although with different levels of student satisfaction and motivation (Luxton-Reilly, et al., 2018).

There is, however, another upstream discussion that has to do with the paradigm to be adopted and which conditions the choice of language. The Curriculum Guidelines for Undergraduate Degree Programs in Computer Science (ACM/IEEE, 2013), a guideline document used by many educational institutions around the world, refers to the growing number of languages being adopted in programming initiation. It also notes that, in the absence of consensus on the choice of programming paradigms and languages, there are institutions whose option is to present more than one paradigm, providing students with a broader perspective, avoiding the focus on the paradigm and language and deconstructing the idea of the existence of the "best" programming language.

Although there is no consensus on the definition of language for teaching, a number of studies clearly identify a set of languages that are most commonly used for this purpose. According to Chalk (Chalk & Fraser, 2006), and the results of a survey to which 44 institutions responded, in 2005 the language most used in education was clearly Java, followed by C++, as can be seen in Chart 2, these data being corroborated by Pears (Pears, et al., 2007).

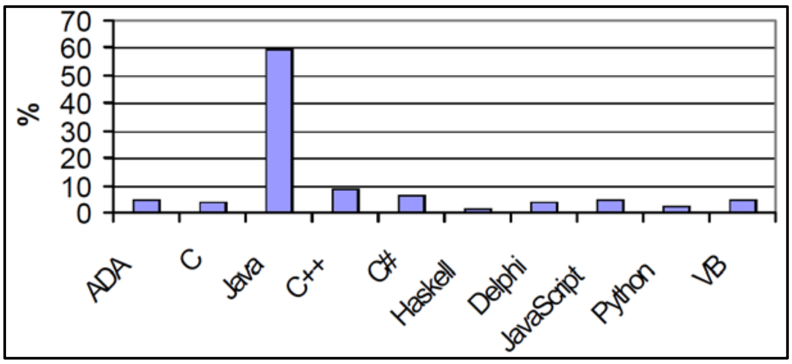


Chart 2 – Percentage of students by programming language

(Chalk & Fraser, 2006)

In 2012, although Java maintained its leading position in terms of popularity, there was a large growth in the use of Python (Siegfried, Greco, Miceli, & Siegfried, 2012).

Even more recently, in a study that analysed courses from 143 universities in 35 European countries, it was concluded that the C language was the most widely used in programming initiation, followed by C++ and Java (Aleksic & Ivanovic, 2016).

Programming language	Semester I	II	III	IV	Σ	Weekly Avg. (Lec+Pra+Lab)
C	155	96	39	20	310	2.2 + 1.2 + 1.1
C++	51	110	34	29	224	2.1 + 1.7 + 1.4
Java	28	54	78	52	212	2.2 + 1.7 + 0.7
C#	5	12	12	21	50	1.8 + 2.0 + 1.3
Pascal	26	12	3	–	41	2.3 + 2.1 + 0.8
Python	19	8	1	1	29	1.9 + 1.2 + 1.0
MatLab	4	11	4	–	19	2.0 + 2.2 + 0.2
Visual Basic	11	2	2	1	16	2.0 + 0.6 + 1.5

Figure 17 – Programming languages at European universities

(Aleksic & Ivanovic, 2016)

There may be some temporal and geographical fluctuations, and the scope and methodology of the studies may also lead to different results. However, it seems unquestionable that C, C++, Java and Python will be the languages most used for the initiation of programming teaching.

2.3 INTEGRATED DEVELOPMENT ENVIRONMENTS (IDE)

For writing the code of a computer program, only a text editor is needed. Taking a Java program as an example, after the written code and the saved file, the program could be compiled and executed using command line tools.

However, the trend has been to develop much more sophisticated systems which, in addition to the publisher, have several other features. These systems are called integrated development environments in that they integrate a set of resources that enhance the programmer's productivity and collaborative work, with less likelihood of errors.

The editor that the IDE provides can usually be configured according to the programming language to be used and the programmer's preferences, and has some help with writing code, such as using color for syntax highlighting, which makes it easier to write and read the code or automatic completion (auto completion) (Satav, Satpathy, & Satao, 2011).

The IDE also provides compilation, execution and debugging tools (debug), and can have several other features such as automatic testing, search facilities, class structuring (in the case of object orientation), project organization, documentation generators, version control or links to databases and external repositories. Many IDEs also allow the insertion of plugins, providing a significant set of new features.

Thus, in general terms, the advantages of using an IDE can be pointed out:

- Increased efficiency, allowing code to be written with less effort and more quickly;
- Integration with other tools such as compiler, debugger, version control system or graphical interface development systems;
- Collaborative work, allowing several programmers to work on the same project, sharing resources and adopting common standards;
- Support for unit testing, integration testing and code coverage;
- Management of software development projects.

However, IDEs may also have some fewer positive aspects, one of the most relevant being their complexity, which may be an additional difficulty causing demotivation, especially for inexperienced programmers.

The set of existing IDEs is large and diverse (Satav, Satpathy, & Satao, 2011), with different origins and purposes. For now, we will only mention Eclipse[49], IntelliJ IDEA[50] and NetBeans[51] as they are the most used by APROG students.

2.3.1 Eclipse

The development of this IDE was started in 2001, by IBM, as an open source platform, and was very popular and widely used by the community. It is a free product and is currently managed by the Eclipse Foundation[52]. Eclipse is a very adaptable and configurable IDE, with a large number of plugins. This gives it great versatility, but the proliferation of plugins from various authors sometimes causes some confusion and difficulty in their choice. Also, the fact that many plugins can be installed may slow down their operation and in extreme cases of incompatibilities between plugins may lead to the IDE having to be reinstalled. Another factor that makes it slow, is the existence of the built-in incremental compiler, since it is executed whenever the code is changed, giving, in real time, indications about eventual errors (Mens, Fernandez-Ramil, & Degrandsart, 2008).

Its organization is based on the concept of workspace allowing different perspectives and visions, making the development environment highly flexible.

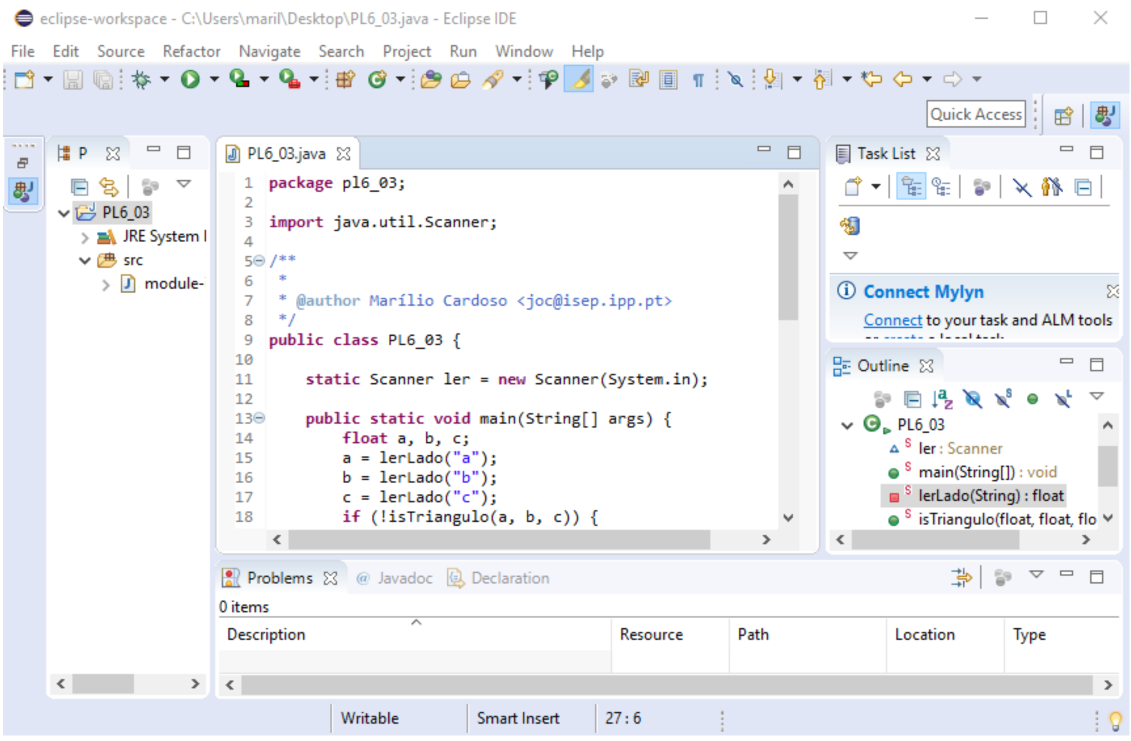


Figure 18 – Eclipse editor's view

49 http://www.eclipse.org/
50 https://www.jetbrains.com/idea/
51 https://netbeans.org/
52 https://www.eclipse.org/org/

In Figure 18 we can see a display of the Eclipse editor.

Eclipse was mostly written in Java, so it is multi-platform, working on several operating systems and supporting several languages including Java, C, C++, PHP and Python. It allows the management of several projects simultaneously, providing tools for analysis and design, testing as well as documentation.

2.3.2 IntelliJ IDEA

IntelliJ IDEA[53] is an IDE developed in 2001 by JetBrains[54], a company based in Prague, Czech Republic, with a commercial version (ultimate edition) and a free (community edition), open source version. This IDE allows encoding in Java, JavaScript, Python, Ruby, Scala and SQL, and several other programming languages (Krochmalski, 2014). The latest enterprise version also features tools for Android application development, as well as duplicate detection functionality and database tools.

JetBrains offers several plugins in its paid version, and there are also others produced by the community for the free version[55].

Its editor has several useful features for the programmer, including inferring the language in which the code is written and giving suggestions for writing as the programmer writes the code. It also has the functionality 'Language Injections', which allows the identification of a language embedded in the code of another such as, for example, SQL in Java.

In Figure 19, the view of ItelliJ IDEA's editor is shown.

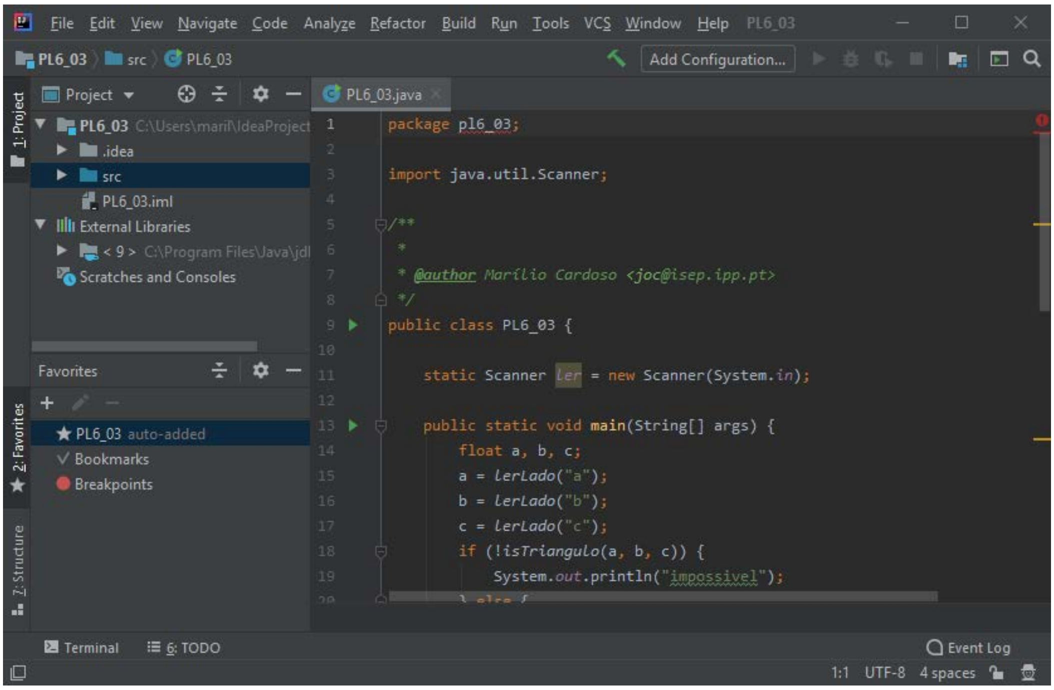


Figure 19 - View from ItelliJ IDEA editor

2.3.3 NetBeans

Just like IntelliJ, NetBeans originated in Prague when, in 1996, a group of university students started developing an IDE for Java that they called Xelfi inspired by Delphi[56]. Later,

53 http://www.jetbrains.com/idea/download/#section=windows
54 http://www.jetbrains.com/
55 http://plugins.jetbrains.com/?idea
56 https://netbeans.org/about/history.html

based on Xelfi, a company was created with the intention of making it a commercial product, and that product was renamed NetBeans.

At that time, Sun Microsystems was developing an IDE for Java, a project that it would abandon by acquiring NetBeans and incorporating it into its product portfolio. The netbeans.org website was released online in June 2000, and NetBeans IDE 3.1 was released in December of the same year. Meanwhile, the decision was made to make NetBeans an open source platform by leveraging increased community input, which led to the appearance of numerous plugins. In 2010, Sun Microsystems was acquired by Oracle (Sharan, 2017) which continued to sponsor the development of NetBeans. Since October 1, 2016, Oracle has ceded control of NetBeans to Apache[57], the latest version being 11.0.

NetBeans is a popular and user-friendly IDE, available for many operating systems and allowing code writing and testing of programs in C, C+, Java, JavaScript, PHP, as well as various other languages (Oracle, 2016). Its advantages are that it is multi-platform, multilingual, free, open source and has several user help resources. However, it can become slow when several projects are opened simultaneously, especially on computers with fewer resources.

The user interface is organised into menus, windows and toolbars offering a configurable, intuitive and functional development environment, as can be seen in Figure 20.

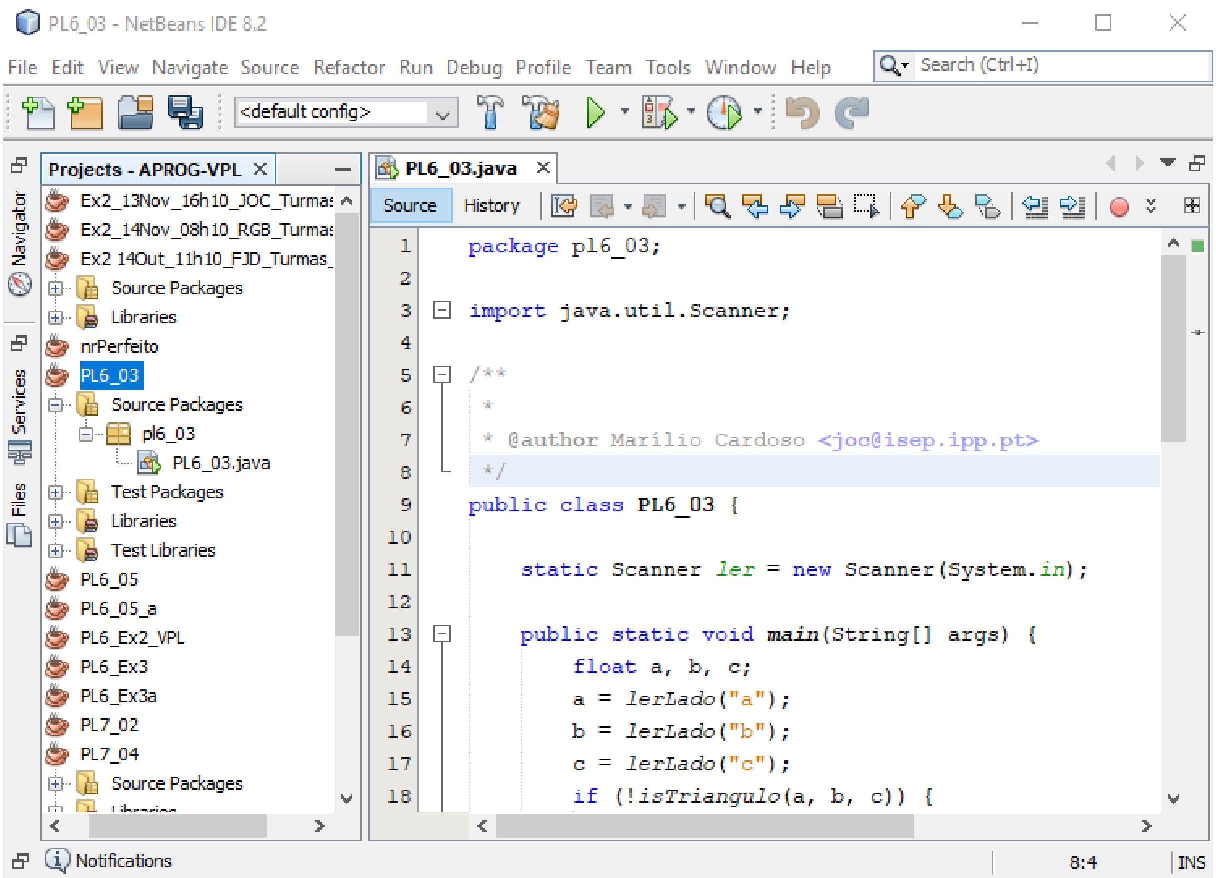


Figure 20 - View from the NetBeans editor

57 http://netbeans.apache.org/

NetBeans supports Maven projects, allows for the management of multiple projects simultaneously, the connection to version control repositories, unit testing tools (JUnit), as well as several other features provided by numerous plugins.

2.4 SOFTWARE DEVELOPMENT MODELS

In the development of any fairly complex product it is necessary to define a strategy and have an implementation plan in order to control resources (financial, material and human) and deadlines.

The development of software is an increasingly important activity, booming with ever larger and more complex projects. A software application can have millions of lines of code, with thousands of methods and a large number of functionalities, interconnecting several modules and promoting the interaction between several systems. Thus, given the size and complexity of many software projects, their development may have to be ensured by several teams simultaneously, sometimes multidisciplinary and geographically dispersed (Guzmán, Ramos, Seco, & Esteban, 2011). This imposes new organisational and working methods, with an increase in collaborative work and with technically competent and, no less importantly, well-developed soft skills (Palacios, Lumbreras, Acosta, García-Peñalvo, & Tovar, 2014). Also, with regard to the maintenance of software, this is not consistent with a code and fix approach.

In the seventies of the last century, the forms of management of the first software projects adopted models of traditional engineering such as civil, naval, mechanical or electrotechnical (Pressman, 2010). These models consisted of activities developed sequentially, with projects defined in great detail and rigor, but not very flexible. This is the commonly called classic life cycle, sequential linear model or cascade model (waterfall) (Royce, 1970).

There are several project management models of software in addition to the aforementioned cascading model, such as rapid prototyping, Rapid Application Development (RAD), incremental development, spiral development (Mishra & Dubey, 2013), and more recently the so-called agile methodologies (Awad, 2005).

2.4.1 Waterfall model

This is a linear sequential model of software development, proposed by Royce in 1970. It originated in industrial processes and construction projects, in highly structured and very inflexible contexts, where it is very difficult to promote changes.

The process consists of a set of phases and develops sequentially, advancing to the next phase only when the current one is finished.

There are some variations as regards the various stages of the process, but the version presented in Figure 21, adapted from (Royce, 1970), is relatively consensual.

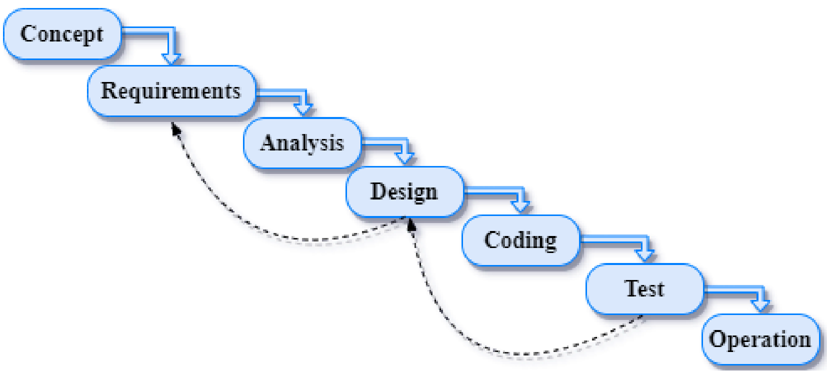


Figure 21 – Process stages in the Waterfall model

The process begins with the concept, with the general idea of what is to be developed, an overall assessment of its interest and feasibility, human resources and estimated costs.

Then, a survey and analysis of functional requirements of the system is carried out on the assumption that they will remain unchanged throughout the project (Palmquist, Lapham, Miller, Chick, & Ozkaya, 2013). At this stage, it is necessary to understand as clearly as possible what the customer's needs and expectations are, and to draw up a software requirements specification document.

In the next stage, the solution is designed according to the requirements and technical solutions, where the data structures, the architecture of software and the user interface models are defined. In the design, algorithms will be developed which will serve as the basis for the encoding, which consists of translating them into a computer language.

After coding, all the components created, individually and in the whole solution, are tested, checking whether they correspond to the desired functionalities for the product. Thus, the tests comprise several levels, from unit tests to system integration and testing. There may be another stage of testing, which are the customer's acceptance tests.

Once any errors found have been corrected, the final version of the product is released, then moving to production. This stage also includes the maintenance and support of the product developed to ensure its operation over time, correcting or adding functionality or complying with requirements arising from legal changes.

All these stages must be fully and rigorously documented, which means that resources and time are required.

This model was dominant until the early 90's of the last century, although there were already several opinions from programmers and researchers that it was not suitable for many of the software projects developed. One such dissenting voice was that of Brooks (Brooks, 1987) when he expressed the conviction that it is not possible to specify a computer application accurately and completely before starting its implementation, an idea supported by Larman when he stated that, unlike other mass production products, software is not something predictable and immune to change (Larman & Basili, 2003). Hence, this model should not be used for the development of large projects, being appropriate only for smaller projects with stable and predictable requirements (Gilb & Susannah, 1988).

2.4.2 Agile Methodologies and Scrum

In several areas of activity, the so-called agile methodologies (Agile) are used for process management and product development (Larman, 2003), and the area of software is a precursor in its use and probably where they are most used (Moniruzzaman & Hossain, 2013). According to Highsmith (Highsmith, 2002), agility refers to the ability of organizations to react to change at a faster pace than changes occur. In this context, there is a paradigm shift, with people playing a central role in the implementation of projects, with soft skills being essential, such as communication, collaboration and motivation.

The agile methodologies are based on a set of 12 principles outlined in the Manifesto for Agile Software Develope[58] of 2001, commonly known as the Agile Manifesto. In addition to these principles, the manifesto defines the guidelines that underpin this approach, with four values being the values of agile development:

- Individuals and interactions more than processes and tools;
- Functional software more than detailed documentation;
- Collaboration with the client rather than contractual negotiation;

[58] http://agilemanifesto.org/

- Responding to change more than merely following a plan.

Agile comprises several practices such as Extreme Programming (XP) (Beck & Andres, 2004), Scrum (Sutherland, 2014) (Cohn, 2013), Kanban (Ahmad, Markkula, & Oivo, 2013), Feature Driven Development (FDD) (Palmer & Felsing, 2002), Lean Development (Poppendieck & Cusumano, 2012), Adaptive Software Development (ASD) (Highsmith, 2000), Dynamic Systems Development Method (DSDM) (Moniruzzaman & Hossain, 2013) or Crystal Clear (Cockburn, 2004).

Of all these, the most popular and most used is surely Scrum (Rubin, 2012) (Kuusinen, Gregory, Sharp, & Barroca, 2017). Scrum was developed by Ken Schwaber and Jeff Sutherland and was publicly presented at the Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) conference, an annual conference of the Association for Computing Machinery (ACM), in 1995, in Austin, Texas, USA. Since then and up to the present, it has continued to evolve and be maintained by its creators, with various contributions over the years resulting from its immense application. Although Scrum is not a complete methodology, as it only defines a structure, it falls within the scope of agile methodologies.

Its name comes from a rugby play, where the players of both teams are grouped in front formations, each playing as a single block, pushing each other to gain possession of the ball.

Scrum is not a standard process in which a set of steps leading to the final product are accomplished, but a framework that allows planning, organizing and managing the development of tasks, based on a set of principles and values that serve as a basis for implementation in each organization. Thus, its use will be appropriate to the reality, interests, practices and specific needs of each organization.

Scrum originates from the empirical theories of process control (empiricism) (Rubin, 2012), which advocate that knowledge results from experience and from making decisions based on what is known, that is, by doing and learning. Thus, it is perceived that there was no replicable model throughout the project as the development team is experimenting, testing solutions, verifying results and adapting methods and techniques in order to improve their performance. The empirical process is particularly suitable for situations of high uncertainty, with the likelihood of rapid and frequent changes.

It is particularly suitable for the development of complex products (Sutherland, 2014) and is widely used in the production and maintenance of software, in an agile, iterative and incremental process. It aims to be a simple, people-centred structure used to enhance teamwork and promote a productive, creative and attractive way of working, based on honesty, autonomy, responsibility, respect, trust and collaboration.

The basis of Scrum operation are consecutive development cycles, called sprints, during which a set of previously defined activities must be performed, but beyond following a plan, Scrum intends to respond to changes.

2.4.2.1 Scrum Pillars and Values

In Scrum, an iterative and incremental approach is used in order to increase risk control and reduce unpredictability, based on three pillars: transparency, inspection and adaptation (Delhij, van Solingen, & Wijnands, 2015).

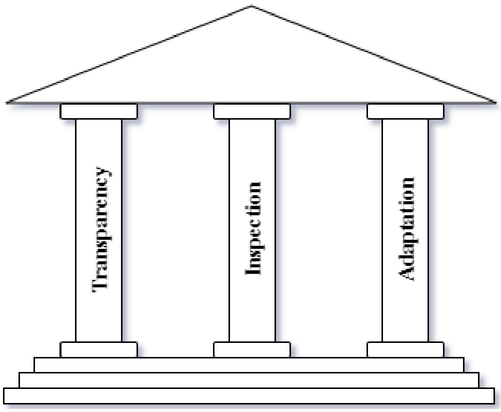


Figure 22 – Scrum pillars

One of Scrum's main advantages is that it provides an overview of any process, allowing continuous monitoring by the team and the possibility of scrutiny by stakeholders. In addition, if difficulties or ambiguities are encountered, they will be easily detectable and amenable to correction and/or overcoming.

The use of Scrum allows for frequent reviews of the project development and the development team's behaviour, evaluating the progress made and the problem-solving capacity.

Transparency allied to inspection leads to adaptation, as it makes the team reflect on its performance, promoting change if necessary and enhancing its capabilities.

Associated to the pillars are Scrum values that are: focus, courage, openness, commitment and respect (Sutherland, 2014). If all team members bear these values in mind and encourage them, they will be more committed, develop their own skills more quickly and the team's objectives will be more easily achievable.

Courage is about doing the right thing and jointly facing the adversities that arise during the development of the project. Each team member should be focused on their specific tasks and be protected by the Scrum Master from all disruptive factors. For the success of the team's work, individual commitment and commitment to everyone's work in a cordial and collaborative atmosphere is fundamental. The team must communicate openly and be willing to examine other perspectives or ideas that arise internally or are presented to them, in an attitude of openness, but consciously and critically.

2.4.2.2 Scrum artifacts

The artifacts are the basic tools to support Scrum and aim to guide the team's action and promote transparency and clarity of information in order to avoid differences in perception of the various elements.

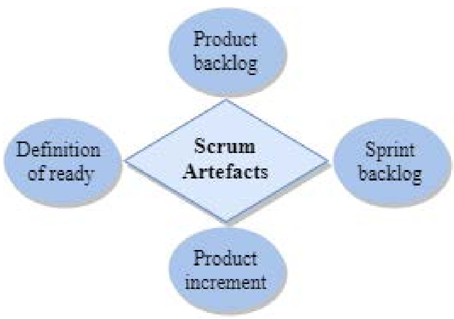


Figure 23 – Scrum artifacts

One of the main artifacts is the product backlog, which consists of specifying functional and desired product requirements through a sorted list (in order of priority) and is considered to be the only source of work from which all planning is defined (Cohn, 2006).

Another artefact is the sprint backlog which, starting from the priority tasks of the product backlog, defines the scope and tasks of the next iteration, estimating the effort and development time. The increment contains all items in the product backlog that were completed during the sprint, along with all increments completed in previous sprints. It is a completed and verifiable work block, constituting a step towards the final goal. It has to be considered complete, respecting the definition of ready.

Although for some authors it is not considered an artifact, the definition of ready-made is an important document since it specifies, in a clear and consensual way for team members and stakeholders directly related to the project, what requirements the increment should have in order to be considered finished.

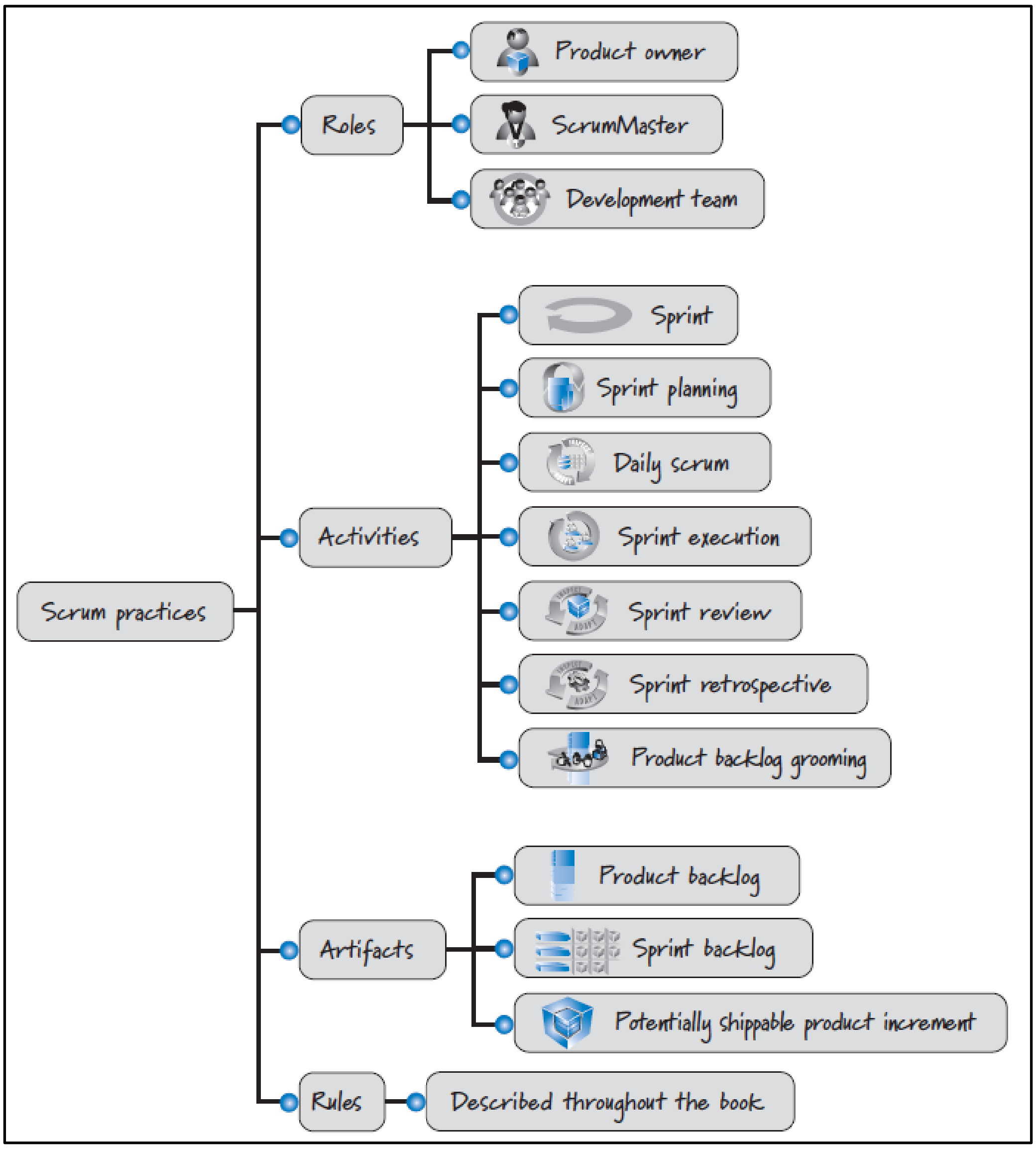


Figure 24 – Scrum practices

(Rubin, 2012)

2.4.2.3 Scrum roles

The development of Scrum-based products consists of one or more small, typically multidisciplinary, highly flexible and adaptive teams, each of which includes actors with three different roles (Sutherland & Schwaber, 2007):

- Product Owner (manages the product);
- Scrum Master (manage processes);
- Development team

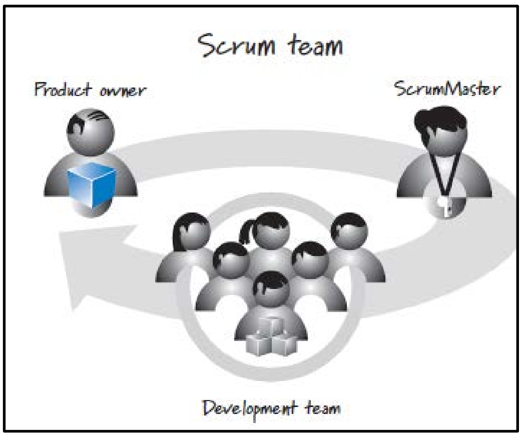


Figure 25 – Scrum roles

(Rubin, 2012)

The Product Owner is responsible for defining the product to be developed and the order in which the tasks will be performed, defining priorities, also being responsible for the overall success of the project. You must represent the customer (or users or other stakeholders), which may be the customer themselves or a representative. It is up to them to have a clear vision of the whole project, communicating it to all the other participants and being always available to provide all the clarifications requested, supported by transparency, in order to allow decisions to be well-founded, thus reducing the risk.

It is up to Product Owner to manage the product backlog in order to maximise deliveries to the customer that represent the greatest possible increase in value, as well as to verify that the work carried out meets the specifications previously defined and to decide whether to accept or reject it. For the success of the project it is essential that the entire structure respects and complies with the decisions of Product Owner.

The management and monitoring of processes and the guidance of the development team are the responsibility of Scrum Master. Its main function is to facilitate and mentor, assisting all those involved (including the Product Owner) to adopt Scrum's principles and values and to refine its use, adapting it to the specificities of the organization and processes. It is also responsible for removing any obstacles that may affect the achievement of results and should protect the team from potential external interference. The Scrum Master is prohibited from issuing technical opinions and their action is important in promoting collaboration and communication among team members and between these and the Product Owner. The Scrum Master is neither the project manager nor the team leader, but an element at the service of the team and its needs.

Scrum contemplates development teams, composed by elements with complementary skills that work collaboratively to obtain a common result. Its function is the development of the product to be delivered to the customer, but also to collaborate with the Product Owner with suggestions that can improve the quality of the final product.

Each team may have seven or more members (Sutherland & Schwaber, 2007), but it should not be too numerous (from 5 to 9) to make it easier to manage and reach consensus on the decisions to be made. It must have a high degree of organisation and autonomy, be involved in planning and have the power to make decisions. The definition of how to deliver the product and the organisation of the team is your own responsibility. Over time, stable teams are associated with higher productivity and it is easier to estimate effort, so it is advisable not to promote frequent changes in team formation.

2.4.2.4 Sprints and ceremonies

The sprints consist of development iterations during which the activities of the sprint backlog should be carried out. It is recommended that all sprints are of equal duration and that they last between two and four weeks. This is due to the understanding that for longer time horizons changes may occur leading to a redefinition of tasks, increasing in complexity and risk, whereas for periods of less than two weeks the work to be done would be too little to enable the production of something significant (and "deliverable").

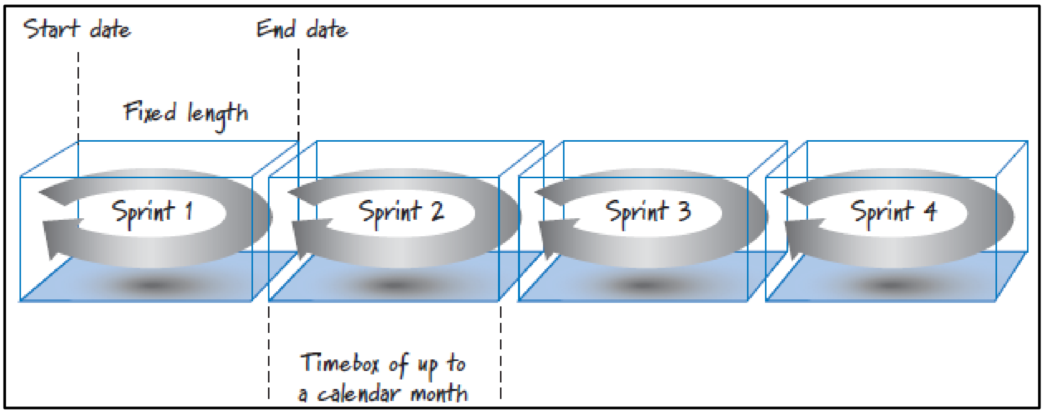


Figure 26 - Sprints

(Rubin, 2012)

In the course of the process, Scrum foresees some practices that it designates as ceremonies and that consist of moments dedicated to specific tasks.

The first ceremony is the definition of the initial scope of the project activities that should result in the preparation of the product backlog.

Each sprint begins after the previous one is completed, starting with its planning, and can be considered as a subproject within the project, with specific objectives and a defined execution time.

At the beginning of each day of work, the team should hold a meeting (daily stand-up meeting) where the work done since the previous meeting will be reviewed, any obstacles or impediments to the normal development of the work will be identified (and removed) and the tasks to be carried out that day defined. In this meeting, each member of the team is to answer three questions:

- What have I done since the last meeting?

- What am I going to do today?
- What obstacles have I encountered that may condition the achievement of the goals?

These meetings should be brief and, for that reason, should be purposefully held with the members of the group standing, so as to cause discomfort and force objectivity. They should always take place at the beginning of the day (or at a fixed time to be agreed by the team members) in order to create regularity in the process. This moment serves to reflect on the work done and highlight the contributions of each of the team members. It is up to the Scrum Master to ensure that it is carried out and that the previously defined duration is scrupulously adhered to.

At the end of each sprint a sprint review meeting (Rubin, 2012) will be held where the team will report on progress made during that period.

To finish, a sprint retrospective will be performed followed by the planning definition of the following sprint.

In Figure 27, the complete development cycle of Scrum can be observed, where the actors and the various ceremonies are listed.

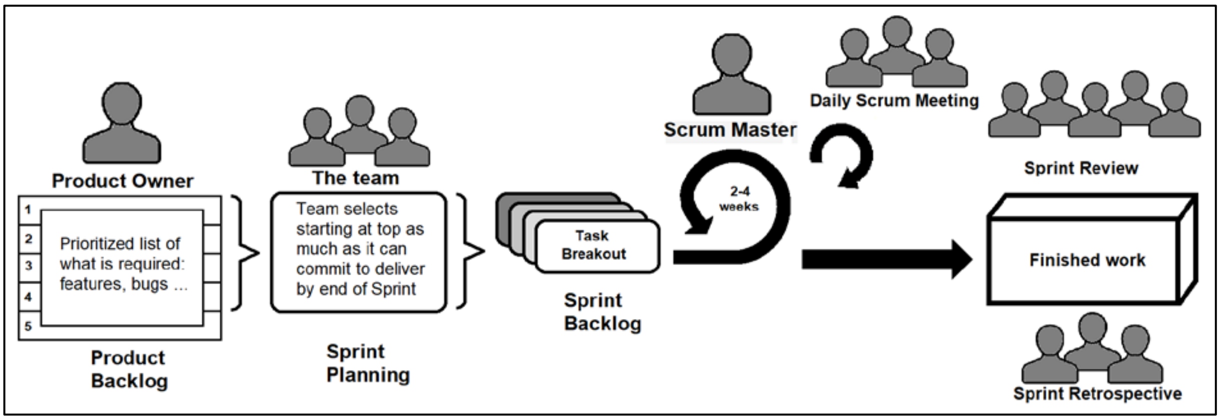


Figure 27 – Scrum development cycle

(Silva, Santos, Angelo, Oliveira, & Moraes, 2016)

2.4.3 EduScrum

In recent years, in the PL-type classroom, eduScrum (Cardoso, Barroso, Castro, & Rocha, 2017) has been used, where students organize themselves into teams for the development of a project. This is a variant of Scrum adapted to education, in which students are responsible for the learning process by delegation from the teacher (Delhij, van Solingen, & Wijnands, 2015).

EduScrum[59], initially applied in secondary education in an experiment that took place in the Netherlands, it has been applied at various levels of education and in contexts where teamwork prevails. Since it is directly derived from Scrum, it presents, essentially, the same characteristics with the necessary adaptations to the specificities of the educational context (Ferreira & Martins, 2016).

EduScrum aims to empower a more intelligent, responsible and enjoyable way of learning (Delhij, van Solingen, & Wijnands, 2015), in a collaborative environment that allows each element to get to know itself better. This way of working fosters autonomy, self-confidence and collaboration, leading to the personal growth of each student and their predisposition and ability to work in groups.

[59]http://eduscrum.nl/

In eduScrum, "what to do" is preferred over "how to do it", challenging students to find their own way of performing tasks in order to make their work profitable. With a strong component of autonomy, it encourages self-organization and critical reflection on one's own actions, enhancing responsibility and maturation.

As with Scrum, eduScrum comprises the same three pillars: transparency, inspection and adaptation.

Transparency requires that the language used is clear and understood by all, that the process is transparent, and that progress is validated under the agreed definition of the finished product. This is also crucial for students to be able to make the most appropriate decisions thus improving the teaching process and maximising the production of value.

Inspection and verification and benchmarking also play an important role in the project development process. The artefacts, tasks and progress achieved must be verifiable and checked frequently in order to identify and correct any deviations, but not with such a frequency that they result in constant interference and an obstacle to the normal course of the work.

If deviations are detected that could jeopardise the achievement of the expected results, they should be corrected as early as possible, with a replanning contemplating corrective measures, thereby promoting the necessary adaptation.

EduScrum recommends six formal events for inspection and adaptation, in an iterative process, from team formation to a personal reflection of each of the elements, which are represented in Figure 28.

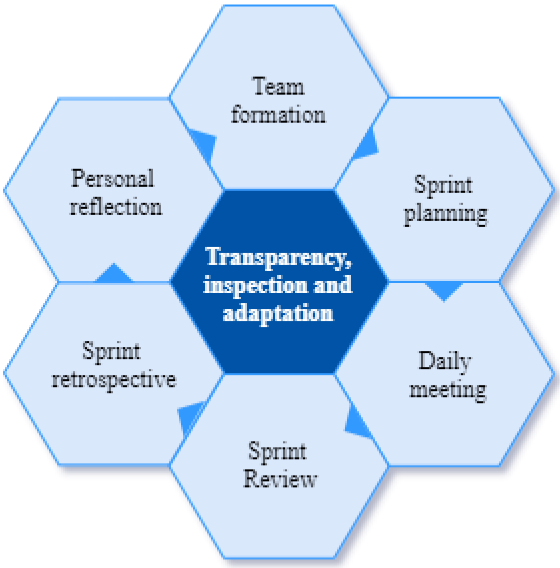


Figure 28 – EduScrum Events

2.4.3.1 EduScrum team

The teams should be small enough to be flexible and easy to manage, but they should integrate sufficient elements to ensure that the tasks are carried out and with differentiated profiles from a multidisciplinary and complementary perspective. EduScrum recommends that a team consists of one teacher (Product Owner) and four students, one of whom plays the role of Scrum Master. However, regarding APROG, as it is an introductory CU, for logistical and operational reasons, the teams include two students only (exceptionally three), and the role of Scrum Master is also taken over by the teacher.

Teams should be autonomous in their organization and in their setting of the Scrum Master. The work of each team is based on autonomy, transparency, self-organization, collaboration, multidisciplinarity and responsibility. Although each team has its own objectives and organisational model, cooperation between teams is encouraged, by sharing experiences and leveraging a benchmarking that can enrich and accelerate the process. The fact that results are delivered in an iterative and incremental manner benefits feedback and the possibility of timely adjustments

2.4.3.2 The roles in eduScrum

The teacher, playing the role of Product Owner, is responsible for defining the purpose of learning, monitoring development and measuring results (Delhij, van Solingen, & Wijnands, 2015). The teacher is also responsible for facilitating the process, by promoting the growth of team members, as well as encouraging collaboration among teams. It is their job to define acceptance criteria and, on the basis of these criteria, to evaluate and assess the results.

The eduScrum Master has a more limited action than the Scrum Master in Scrum. This is due to the fact that students are usually still inexperienced, with the teacher performing some roles that are typically the responsibility of the Scrum Master.

The eduScrum Master is responsible for managing the board, a document in which the activities and status of each of them are recorded. This is an important and useful document for the development of the project, allowing the identifying of responsibilities and the study of tasks development.

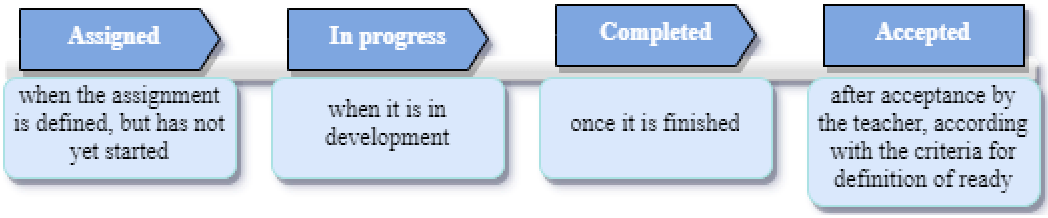


Figure 29 - Status of activities on the board

Figure 29 shows the different states in which each of the activities can be recorded on the board.

3 TEACHING-LEARNING PROCESS OF PROGRAMMING

"By far the best proof is experience"

Francis Bacon

This chapter aims to address issues related to teaching and learning programming.

Some particular aspects of the teaching-learning process of programming are highlighted, with references to problem solving and the act of learning and teaching how to program.

A reflection on algorithms and logical reasoning is made, clarifying concepts and representation techniques such as the pseudocode and the flow chart. The matter of verification of functionality by tracing and the use of tools for this purpose is also addressed.

Some concepts are also presented regarding the types of learning and the forms of face-to-face and distributed teaching.

Finally, aspects related to the use of learning management systems in general are addressed, and some examples of LMS with emphasis on Moodle are mentioned.

3.1 PROBLEM SOLVING

Programming can be defined as the act of perceiving a problem, formulating a solution and implementing it by translating it into a programming language so that it can be "understood" by a computer.

Throughout history several authors have addressed the issue of problem solving and the inherent stages of this process. As early as the 17th century, Descartes, in his book Discourse on Method (Descartes, 2008), recommended four basic precepts for this purpose:

- Not to accept something as true unless it was unequivocally known as such, avoiding any premature or hasty judgment;
- Divide each of the difficulties that arise, in as many parts as possible, in order to facilitate their resolution;
- Sort out thoughts, starting with the simplest and easiest to understand and moving gradually towards the most elaborate and complex;
- Carry out thorough listings and reviews so thoroughly that nothing is omitted.

The most recent theories, emanating from the context of mathematics and information processing, advocate models that, in their genesis, follow principles similar to those stated by Descartes. An example is the Identification, Definition, Exploration, Action, Learn model (IDEAL) (Bransford & Stein, 1984) or the set of techniques and methods synthesized in the acronym Focus, Analysis, Resolution, Execution (FARE) (Santucci, 2010).

Within the framework of programming, problem solving is carried out by methods based on sequential steps that obey a certain logic (Ochse, 1990), whose more synthetic formulation comprises the following steps:

- Data input, where the analysis is carried out and the problem is understood;
- Processing, when designing and evaluating alternatives and selecting the one to be implemented;

- Output, a stage that includes the planning and implementation of the solution;
- Review, where the solution is evaluated, and any necessary corrections or changes are made.

The Organisation for Economic Co-operation and Development (OECD) implemented an international student assessment programme in 2000, which aims to measure competence in reading, mathematics and science as well as financial literacy and problem solving. The programme has been named Programme for International Student Assessment[60] (PISA), it is updated every three years and is still in force today, with reports being published on various subjects and different perspectives. The PISA 2012 Assessment and Analytical Framework (OECD, 2013) report mentions a study on problem-solving skills, showing the importance given to this specific skill by the different curricula in different countries.

According to Lesh & Zawojewski, problem-solving capacity is fundamental for the development and consolidation of future (Lesh & Zawojewski, 2007) learning , as well as for the realization of personal activities and full and effective participation in society.

Based on the literature and OECD studies, Gomes identified the steps to follow in order to solve a problem and the main difficulties experienced by students (Gomes A. , 2010):

- Understanding the problem - It is a fundamental step in achieving the goal although it is often overlooked or undervalued. It is common during the implementation of the solution for the student to become aware that, in fact, they had not understood the problem in a solid and complete way, which will inevitably imply a step back in the process, bringing them back to an initial stage.
- Characterization - It refers to the way variables are identified, the relationships between them and the degree of importance of each one. Characterization also includes research and analysis of analogous situations, allowing the study of solutions to similar problems.
- Representation - It is at this stage that verbal representations are developed, in the form of text, graphs or tables that translate the students' understanding of how to solve the problem.
- Resolution - For the overall resolution it is necessary to integrate the various parts that make up the problem, because in order to solve the problem, one of the most used strategies is to break it down into smaller and simpler tasks (top-down approach), solving and refining each of them and bringing the solutions together to form a coherent whole.
- Reflection on the solution - Usually the student's objective is only to find a satisfactory solution, not to reflect on it, not trying to refine or optimize it. However, the analysis of the work done, the search to improve the solution and the identification of alternative solutions is of utmost importance for the consolidation of programming skills. The various solutions developed by different learners or groups can be discussed together, by sharing points of view, difficulties and resolution strategies, questioning options and fostering discussion and enriching the learning process.
- Communication of the solution to the problem - The presentation of the solution to elements external to the process requires a deeper reflection and a more careful selection of the means used, providing new questions and possibilities for improvement.

As each of the stages is obviously important, the first stage (understanding) is highlighted as fundamental, as ambiguities and misunderstandings inevitably lead to an unsatisfactory solution. Reflection on the solution and its communication is also a practice used in teaching

[60] http://www.oecd.org/pisa/

programming at ISEP and has proved to be important and effective in sharing experiences and different approaches, contributing to the consolidation of learning.

3.2 TEACHING AND LEARNING TO PROGRAM

Teaching programming is one of the most challenging tasks at any level of teaching, being a complex task for any teacher or instructor (Koulouri, Lauria, & Macredie, 2015), but especially for those who have beginning students. From the student's perspective it is usually viewed with some apprehension because learning to program is different from acquiring other kind of knowledge. Thus, according to Moström (Moström, 2011), this can be a very difficult process for students, requiring multiple skills such as: understanding, memorization, problem solving capacity, abstraction and logical reasoning, among others (Piteira & Costa, 2013).

Learning to program is not a process in which formulas of any physical-mathematical theory are applied, nor is it based on memorization (Fuentes-Rosado & Moo-Medina, 2017). For this purpose it is not enough to simply know the syntax of the language, it is necessary to have the ability to solve problems and translate a solution into a programming language (Gomes & Mendes, 2007). Each problem can have more than one way of being solved and each programmer can have styles and approaches that make the solution different from others. This is the main difficulty associated with teaching and learning programming (Fuentes-Rosado & Moo-Medina, 2017).

To program it is necessary to have the solution design in advance. Learning to design the solution without the means to test it becomes boring and ineffective. Likewise, learning the syntax of a language without having a problem and the design of its solution does not allow the training of what is being learnt. Thus, the way to solve the issue is by learning, in a sequential and iterative way, to solve the problem and to implement the solution, codifying it.

As the teaching of programming is a complex and demanding process, it must be carried out in an iterative and incremental manner. One of the most used ways to implement it is the introduction of exercises of increasing intensity and difficulty, which provide the student with activities that are sufficiently challenging but, at the same time, feasible, in order to maintain motivation and interest, allowing the evolution of learning (Gomes A. , 2010).

Motivation is an important factor in human activities, and personal satisfaction and achievement are basic aspects for individual success, as recommended by the referred to Theories of Satisfaction (Campos & Ramos, 2013). Of these, one of the most well-known is Maslow's Theory of the Hierarchy of Needs, which, coming from the world of psychology, is particularly famous in the field of management, in particular for the designated pyramid of needs (Maslow, 1987) (Figure 30).

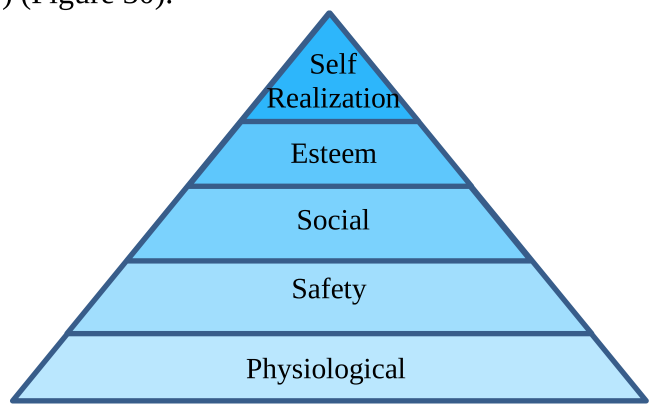


Figure 30 – Maslow's Pyramid of Needs

(Chiavenato, 2003)

This pyramid represents the called hierarchy of human motivations that distinguishes between basic needs, such as physiological needs, and higher-level needs, such as needs for relationships or personal fulfilment. According to Maslow a need when satisfied will be replaced by that of the next higher level (Maslow, 1987).

Learning is related to the satisfaction of high-level needs, and therefore implies meeting the needs of lower levels, which is not always entirely true. As a result, in addition to the difficulties intrinsic to teaching, the teacher is sometimes confronted with other situations that complicate their actions and require closer monitoring of the students.

In recent years there has been a concern to initiate the teaching of programming at an earlier stage. According to several authors, learning programming by children increases their logical reasoning, improves cognitive abilities and human interactivity and transforms perception to establish logical connections (Bers, Flannery, Kazakoff, & Sullivan, 2014), also contributing to self-confidence and problem solving capacity.

Even for those who do not work or intend to work in the ICT area, programming skills can be important for their personal development as well as for their professional activity, even if it is not directly related to the IT area. Contact with programming improves the ability to solve problems, enhances better understanding of the context of technology facilitating automation and optimization of tasks (Kalelioğlu, 2015).

3.3 ALGORITHMS AND LOGICAL REASONING

The word algorithm is derived from the name of the Persian mathematician Mohammed Al-Khwarizmi (Brezina, 2006) and he is the author of the rules for the elementary operations of arithmetic. His most relevant work was the book Al-jabr wal-muquabalah which gave rise to the development of algebra and algorithms (Vasconcelos & Carvalho, 2005). One of the oldest and most famous algorithms in the world is the Euclidean algorithm that serves to determine the maximum common divisor of two whole numbers.

The word logic comes from the Greek term logiké and is associated with philosophy and mathematics. In the Portuguese language online Priberam[61] logic is defined as the science of reasoning, being also often associated with coherence, method and systematization.

It is usual that the process of teaching programming begins with logical reasoning and algorithms and, at a later stage, knowledge of the logic learned will be applied using a specific programming language (Cardoso, Castro, & Rocha, 2018). The algorithm is the basis of the process allowing to define solutions in a conceptual way.

Developing algorithms means defining a finite and well-defined set of clear and unambiguous rules to solve a problem in a finite time interval and should be effective and efficient. Thus, an algorithm can be defined as a sequence of steps that accepts input values and leads to a value or set of values as output (Cormen, Leiserson, Rivest, & Stein, 2009), or, as Skiena states (Skiena, 2008), the idea behind any reasonable computer program.

The algorithm can also be seen as a tool for solving a well-specified computational problem, which describes the specific computational procedure for obtaining the relationship between input and output data, as defined in the problem to be solved (Cormen, Leiserson, Rivest, & Stein, 2009).

According to Knuth, an algorithm should hopefully have the following characteristics (Knuth, 1997):

- Definition - Each step must be clearly defined, with the specification of the actions to be carried out being made in a rigorous and unambiguous manner;

[61] https://dicionario.priberam.org/

- Finitude - An algorithm has a finite number of steps, and must end after all of them have been executed;
- Input - Data that is "provided" to the algorithm at the beginning or throughout the execution process;
- Output - Values that result from the execution of the various instructions, depending on the input data;
- Effectiveness - The specified steps should lead to the resolution of the problem, in a finite time and with a finite amount of effort, and the instructions should be simple enough to be performed by a human, using only paper and pencil.

Associated to the algorithm we can find the logic and data structure that will result in the implementation of the solution in a given programming language (Figure 31).

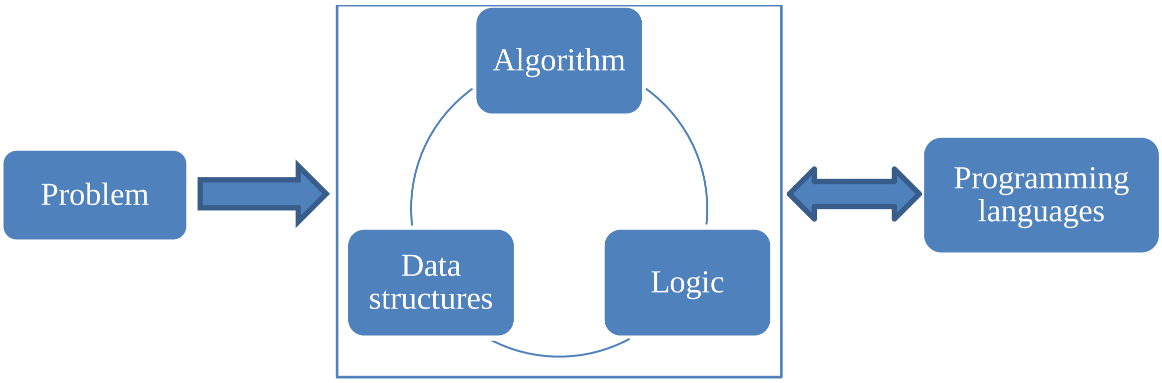


Figure 31 – Algorithm

An algorithm can be expressed in natural language in any language such as Portuguese or English. However, natural language tends to be too descriptive and ambiguous, with several words having similar meanings to each other and others that may have context-dependent meanings. Thus, the resolution of a problem must be presented in a synthetic, clear, unambiguous and easily codable form, i.e. translated into a programming language. For this purpose, pseudocode and/or flow charts are used, culminating the process with a verification mechanism usually referred to as tracing.

3.3.1 Pseudocode

Pseudocode consists of a textual form of algorithm representation, with its own very simple syntax, following a certain convention. It presents a more restricted vocabulary than the narrative description in a natural language (Cormen, Leiserson, Rivest, & Stein, 2009), thus decreasing the probability of occurrence of ambiguities and misunderstandings, and facilitating the explanation of the logic associated with the algorithm. This formalism is intended to allow the programmer to focus on the logic of the process and to abstract from the specific syntactic aspects of each programming language.

Within the pseudocode it is usual to define primitives, originating from the language used, which allow specifying basic instructions typically used in programming languages, such as reading data, assigning values to variables or writing results. In the case of Portuguese these primitives can be, for example: "*LER*" (READ), "*ESCREVER*" (WRITE) and "*SE*" (IF), for reading data, writing results and as condition instruction respectively. However, as English is an extraordinarily widespread language used internationally, and in particular in the field of computing, there are many algorithms developed with elementary instructions in English. In addition, most programming languages use instructions that derive directly from the English language.

Pseudocode is an intermediate form between natural language and a programming language, using words, usually from the programmer's native language, that have equivalents in the programming languages, facilitating their coding.

Although in pseudocode there is no rigid notation, some elements that serve as a basis for codification are commonly accepted, which are presented in Table 4.

Table 4 – Pseudocode elements

Structure	Elements
Data	Variables and constants
Data types	Integer numbers, real numbers, characters, character chains, logical, indexed
Operators	Arithmetic, logical, relational
Instructions	Assignment, input, output
Control structures	Sequence, decision, repetition

In most programming languages it is necessary to define the data structures that will allow the data to be stored during programme execution. Therefore, also in the writing of the algorithm in pseudocode the variables necessary for the operation of the program should be identified and characterized in order to facilitate encoding.

As an example, a problem is presented for pseudocode resolution:

"Given an integer number the programme should determine and present its absolute value"

```
ALGORITM absolute value
DS: value, result INTEGER
START
    WRITE ("Insert an integer value")
    READ(value)
    IF (value >= 0) THEN
        result ← value
    ELSE
        result ← value * -1
    ENDIF
    WRITE("The absolute value of ", value, " is ", result)
END
```

Figure 32 – Example of solving a problem using pseudocode

An algorithm written in pseudocode must follow a structure, as shown in the example, with its name after the word "*ALGORITM"*, and the variables to be used after the data structure is specified ("*DS*:"). In addition to these elements, the body of the algorithm is started by "*START*" and ended by "*END*", on which all instructions will be written, each of which must be written on a separate line.

In addition to the above-mentioned input and output instructions ("*READ"* and "*WRITE"*) it is possible to call up several others relating to conditional structures, repeating structures, arithmetic expressions and logical expressions, making it possible to specify more complex algorithms.

In the writing of the algorithm there are also other precautions that must be observed, in order to facilitate its reading and interpretation, helping in the "translation" into the chosen programming language, such as indentation and writing comments.

3.3.2 Flowchart

A flowchart, or flow diagram, consists of a description of the flow of a process or algorithm in graphical form. Since there is no consensus as to who is to be attributed the authorship of the flowcharts, it is believed that the concept was first used in 1921 by Frank Gilberth in the presentation of the work "Process Charts. First Steps in Finding the one best way to do work.” at an American Society of Mechanical Engineers conference (ASME) (von Rosing, Scheer, & von Scheel, 2015). Later, in the 1930s, the industrial engineer Allan H. Mogensen used the Gilberth principle for the creation of process diagrams, in a more elaborate way, which allowed the concept to be disseminated. However, the generalization and intensification of the use of flowcharts occurred from 1947, when ASME defined an international standard of process symbols (ASME, 1947) and established how to use them in graphs and diagrams (Graham, 2004). Since then, flowcharts have often been used to define algorithms, but they are an effective tool in several other fields of application.

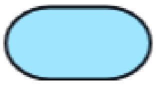
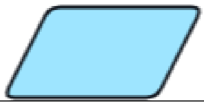
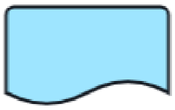
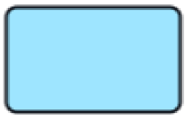
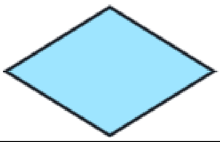
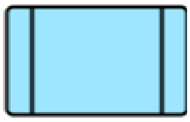
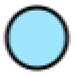
A flowchart is a type of diagram that allows describing processes, consisting of a reduced set of graphic elements linked together by unidirectional arrows, thus allowing the process flow to be explicitly identified, with the steps leading to the resolution of the problem. In each flow chart, the flow is usually shown from top to bottom and from left to right. The elements represent steps in the process allowing a clear visual perception with succinct information, allowing the demonstration of the logical reasoning behind the solution elaboration.

Graphic representation has the advantage of being visual, providing a more immediate and intuitive perception because, as Pressman states, "an image is worth a thousand words", adding, nevertheless, that it is very important to know which image and which thousand words (Pressman, 2010), also alerting to the potential errors arising from a misuse of graphic tools.

The graphical elements used in algorithms are typified in standards, the most relevant being ISO 5807-1985 (Information processing - Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts). This standard defines the symbology and the rules for drawing up flowcharts for the development area of software and results from two previous standards: ISO 1028:1973 (Information processing – Flowchart symbols) and ISO 2636:1973 (Information processing – Conventions for incorporating flowchart symbols in flowcharts). However, the origin of standardisation in this context dates back to 1970, with the publication of ANSI X3.5.

Table 5 shows the most commonly used symbols for flow charts, defined in ISO 5807-1985.

Table 5 - Flowchart symbols[62]

Symbol	Name	Meaning
	Terminal	Definition of beginning and end of the logical flow of a program
	Input	It represents data entry, by any type of medium or device, although usually associated with the keyboard
	Output	It represents the execution of the data output operation in a printed document, although its use as data output to any device has become widespread
	The process	Representation of the execution of an operation or set of operations that establish the result of a logical or mathematical operation
	Decision	It represents the use of conditional deviations for other points in the program according to varying situations
	Module / subroutine	It represents the definition of a group of operations established as a processing subroutine
	Flow line	It shows the direction of the process Each flow line connects two elements
	Link	It represents the input or output of another part of the diagram

The use of flowcharts for the initial teaching of programming has proved to be useful, as it allows the logic to be made explicit, without revealing the specific implementation details of each language, allowing the student to focus only on the conceptual resolution of the problem.

To illustrate the use of a flowchart, we will refer to the example previously presented and already implemented in pseudocode: “Given an integer the program should determine and present its absolute value” (Figure 33).

[62] https://www.iso.org/standard/11955.html

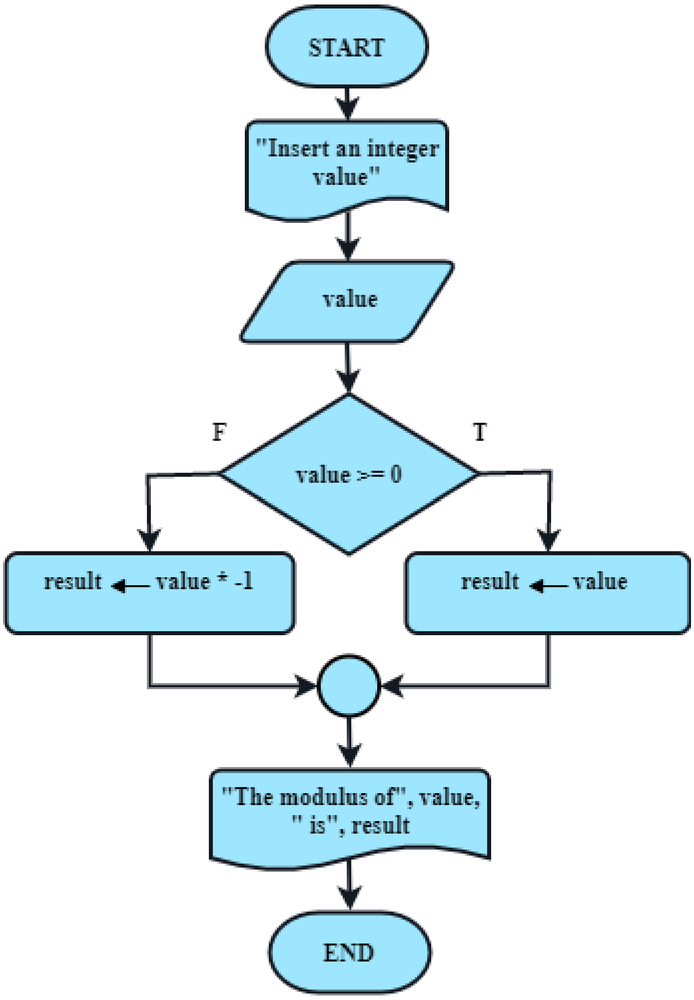


Figure 33 - Example of a problem resolution using a flowchart

As with the pseudocode, also in the flowchart the algorithm should have only a beginning and an end, which should be clearly represented. Since the flowchart is the graphic representation of the algorithm and can be directly transposed to the writing of the code, it must contain the elements that make this translation possible in a simple and unequivocal way.

3.3.3 Tracing

After writing the algorithm (either in pseudocode or flowchart) its functionality must be verified using a test process. In Portuguese language this verification process is commonly referred to as "traçagem", in a free and abusive translation of trace. It should probably be called tracking, but the term has become widespread and is now used and fully accepted in the context of algorithm writing. There is also another term related to error detection and correction, called debugging, which is associated with errors that may appear in the code written in a particular programming language. The errors can be of a syntactic and logical nature. In pseudocode encoding, the relevant verification is in the scope of logic, and syntactical errors are not considered, since the pseudocode does not have a syntax itself, but rather some conventions and guidelines. Thus, the tracing process consists of testing the algorithm, in a step-by-step execution simulation, for certain input values, previously defined, and for which the expected result is known. It is then necessary to define a set of input values, comprehensive enough to be able to check the internal behaviour of the algorithm along its steps, and for the potentially different situations it may be subjected to. This process of identifying the various possibilities and defining the values to be tested, prior to the execution of the tracing, corresponds to the preparation of a test plan. As an example, a test plan for the algorithm of the example already presented of the determination of the absolute value of a number is presented in Table 6.

Table 6 – Test plan example

Test	Input	Expected outcome	Result achieved	Check
1	value = 10	The absolute value of 10 is 10		
2	value = -5	The absolute value of -5 is 5		

For the implementation of the tracing it is convenient to construct a table where the entities that can vary throughout the implementation process, namely the variables and the conditions, are entered. Just as in the development of algorithms, there are no absolute rules when it comes to tracing, and there may be some variants in their development. For beginner students and with simple algorithms it is recommended that the table be constructed in the same plane as the pseudocode, as in the example presented in Figure 34 for the problem of the absolute value of a number.

	Tracing	
ALGORITHM absolute value		
DS: number, result INTEGER		
START		
WRITE(“Insert an integer value”)		
READ(value)	value = 10	value = -5
IF (value >= 0) THEN	true	false
result ← value	result = 10	
ELSE		
result ← value* -1		result = 5
ENDIF		
WRITE(“The absolute value of ", value, " is ", result)	The absolute value of 10 is 10	The absolute value of -5 is 5
END		

Figure 34 – Algorithm example and its tracing

After the tracing has been performed, it is then possible to check the functioning of the algorithm, for the defined input values, by completing the table relating to the test plan, as shown in Table 7.

Table 7 – Verification algorithm example after tracing

Test	Input	Expected outcome	Result achieved	Check
1	number = 10	The absolute value of 10 is 10	The absolute value of 10 is 10	OK
2	number = -5	The absolute value of -5 is 5	The absolute value of -5 is 5	OK

It should be noted that tracing only allows checking whether the algorithm is wrong, detecting the presence of errors, for the values tested. However, even if it works for all arbitrary input values, it cannot guarantee the absence of errors, thus not ensuring that the algorithm works correctly for other input values (Vasconcelos & Carvalho, 2005).

For reasons of simplicity the example given, by way of illustration, contains only decision structures. However, in the overwhelming majority of processes the use of other control structures, in particular repeating structures, will be necessary. In such cases, the strategy of tracing along the code may be impracticable, alternatively, a table where the state of the various variables and conditions is recorded vertically, as exemplified in Table 8, for the example of the absolute value can be used.

Table 8 – Tracing example

number	IF (value>= 0)	result	Output
10	true	10	The absolute value of 10 is 10
-5	false	5	The absolute value of -5 is 5

To illustrate the use of tracing in a process with repeat structures let us consider the example of the factoring calculation of a user-defined positive integer. In this case an iterative solution will be used, so knowing the number of iterations to be performed, a counted repetition structure will be used ("FOR").

```
ALGORITM factor
DS: value, i, result INTEGER
START
    WRITE("Please enter a positive integer number")
    READ(number)
    IF (number< 0) THEN
        WRITE("Error! The number must be a positive integer")
    ELSE
        result ← 1
        FOR i ← 1 TO number STEP 1
            result ← result * i
        ENDFOR
        WRITE("The factorial of ", number, " is ", result)
    ENDIF
END
```

Figure 35 – Algorithm for calculating the factorial number

The number of iterations to be performed will be equal to the number for which the factorial is intended to be calculated. If, for example, the factorial of number 6 were to be calculated, six iterations would be performed, which, in a "row" tracing with the code, as exemplified in Figure 34, would correspond to six columns. In Table 9 an example of the algorithm plotting of Figure 35 is shown, with iterations per line.

Table 9 - Tracing of the algorithm example for the calculation of the factoring

number	IF (number< 0)	result	i	i <= number	Output
6	false	1	1	true	
		1	2	true	
		2	3	true	
		6	4	true	
		24	5	true	
		120	6	true	
		720	7	false	The factorial of 6 is 720

3.3.4 Portugol and VisuAlg

As previously mentioned, when writing an algorithm in pseudocode, the primitives used usually originate from the programmer's native language. Thus, for Portuguese speakers there is a Portuguese version of pseudocode, widely used in the introduction to programming. This version is known as "Structured Portuguese", "Algorithmic Portuguese".

The Portuguese Algorithmic allows the writing of algorithms, in Portuguese, in a simple and intuitive way, its main field of application being the teaching of logic and algorithms. At the origin of its conception was the idea of simplicity, so as not to require experience or in-depth knowledge for its use. The objective is that the focus is on the process and not on the syntax, fostering abstraction and logical reasoning (Noschang, Pelz, Jesus, & Raabe, 2014).

Portugol is a pseudo language, compilable and executable, so its use, whilst taking some freedom in writing from the algorithm to the extent that it requires a stricter notation, has the advantage of facilitating tests of its functionality by using tools for that purpose.

There are several interpreters and compilers that, although presenting some differences in syntax and mode of operation, are based on Portugol. By way of example the following can be cited: CompAlg (Bilabila, 2017), Portugol IDE[63] (Manso, Marques, & Dias, 2010), Portugol Studio[64] (Noschang, Pelz, Jesus, & Raabe, 2014), Portugol Viana[65] (Cruz, Cerqueira, & Gradíssimo, 2009) and VisuAlg[66] (Leite, Senefonte, Barbosa, & Seabra, 2013).

One of the tools to support the development of algorithms that was possible to explore throughout this study was VisuAlg. This is a free software that allows the creation, editing, interpretation and execution of algorithms in Portugol.

VisuAlg was developed with the aim of helping students in a seemingly unattractive subject where they usually experience some difficulties, providing them with a way to test their work in a more stimulating way than traditional paper tracing (Almeida, 2013).

VisuAlg has a set of features that range from writing the algorithm in an editor to the possibility of executing it step by step by debugging the pseudocode.

According to Souza (Souza, 2009), in VisuAlg it is possible:

- the execution of the code step by step;
- the visualization of the contents of the variables;
- the analysis of the status of the activation stack in the case of subprograms;
- to have a run counter for each program line.

[63] http://orion.ipt.pt/~manso/Portugol/download.html
[64] http://lite.acad.univali.br/portugol/
[65] https://sourceforge.net/projects/portugolviana/
[66] http://visualg3.com.br/

Tools of this type are being increasingly adopted because they improve the effectiveness of the traditional method of algorithm development and verification and give the student the possibility to record the work and continue working on it later.

A study on the use of VisuAlg was conducted at the State University of Londrina, Brazil, in which students from the first year of the Computer Science course in the subject of Programming Techniques (Leite, Senefonte, Barbosa, & Seabra, 2013) participated. The results of the study led to the conclusion that VisuAlg is a very useful tool for teaching algorithmic concepts, and that learning has become more practical, faster and enjoyable.

The fact that the tool separates the codification of the state of the variables and the results, has a positive impact on the students' perception, allowing them to better learn the concepts and, by executing step by step, "see" the result of each instruction.

To illustrate some aspects of the operation of VisuAlg, we will use the example of the calculation of the absolute value of a number, and you can see in Figure 36 the general appearance of the tool environment, with the option menu, command bar and the different zones.

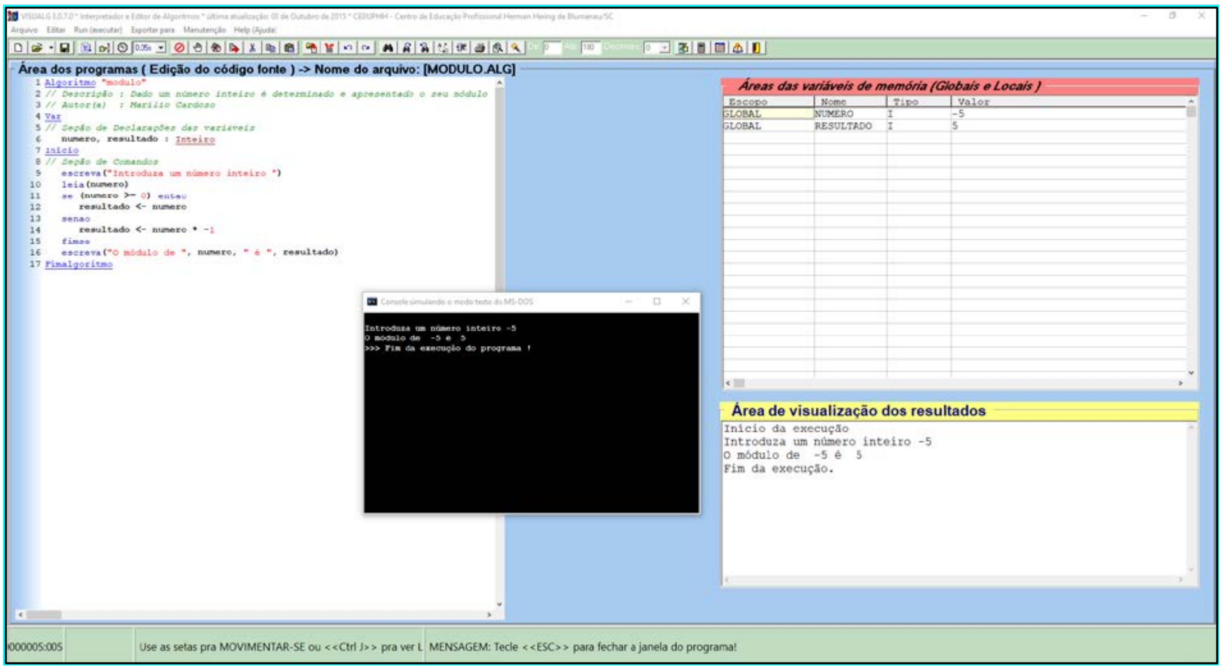


Figure 36 – Overview of VisuAlg

When a new file is created, a template with the basic structure is automatically presented in the algorithm area, as can be seen in Figure 37, allowing you to make changes and write the pseudocode.

Área dos algoritmos (Edição do código fonte) -> Nome do arquivo: [semnome]

```
Algoritmo "semnome"
// Disciplina  :  [Linguagem e Lógica de Programação]
// Professor   : Antonio Carlos Nicolodi
// Descrição   : Aqui você descreve o que o programa faz! (função)
// Autor(a)    : Nome do(a) aluno(a)
// Data atual  : 28/09/2019
Var
// Seção de Declarações das variáveis


Inicio
// Seção de Comandos, procedimento, funções, operadores, etc...


Fimalgoritmo
```

Figure 37 - Pseudocode template

In Figure 38 the example of the calculation of the absolute value of a number, in Portugol, is presented, and the one previously stated regarding the lesser flexibility in writing should be highlighted. It should be noted that previously the primitive "RSI" was used for data entry, just as another term with similar meaning could have been used. However, in VisuAlg, the reading instruction is exclusively "read", but can be written entirely in upper or lower case or even with some letters in upper or lower case, because, unlike the most commonly used programming languages, it is not case sensitive. Also with regard to the output instruction, the commonly used primitive is "WRITE". However, in Portugal, the instruction "escreva" ("write") or the instruction "escreval" can be used, the latter being used to change the line after writing.

Área dos programas (Edição do código fonte) -> Nome do arquivo: [MODULO.

```
Algoritmo "modulo"
// Descrição : Dado um número inteiro é determinado e apresentado o seu módulo
// Autor(a)  : Marílio Cardoso
Var
// Seção de Declarações das variáveis
   numero, resultado : Inteiro
Inicio
// Seção de Comandos
   escreva("Introduza um número inteiro ")
   leia(numero)
   se (numero >= 0) entao
      resultado <- numero
   senao
      resultado <- numero * -1
   fimse
   escreva("O módulo de ", numero, " é ", resultado)
Fimalgoritmo
```

Figure 38 - Example of an algorithm in Portugol, VisuAlg

As shown in the top right corner of Figure 36, VisulAlg provides information with the complete characterization of each variable, and it is possible to observe whether it is local or global, the name, the type and its content, as can be seen in more detail in Figure 39.

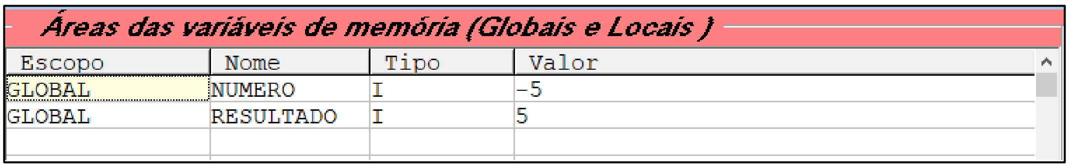


Figure 39 - Variable areas in VisuAlg

In Figure 40 a view of the area where the results of the algorithm execution are presented is also shown.

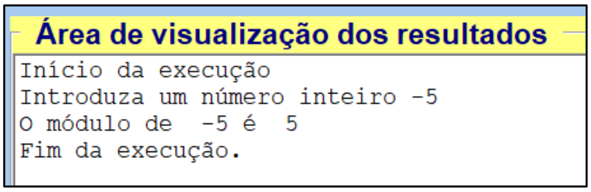


Figure 40 - VisuAlg results display area

In the "step-by-step" execution, the line that is being executed is marked at each instant, as can be seen in Figure 41. Note that, after entering the number, the instruction that is to be executed next is the one on line 11, which is highlighted.

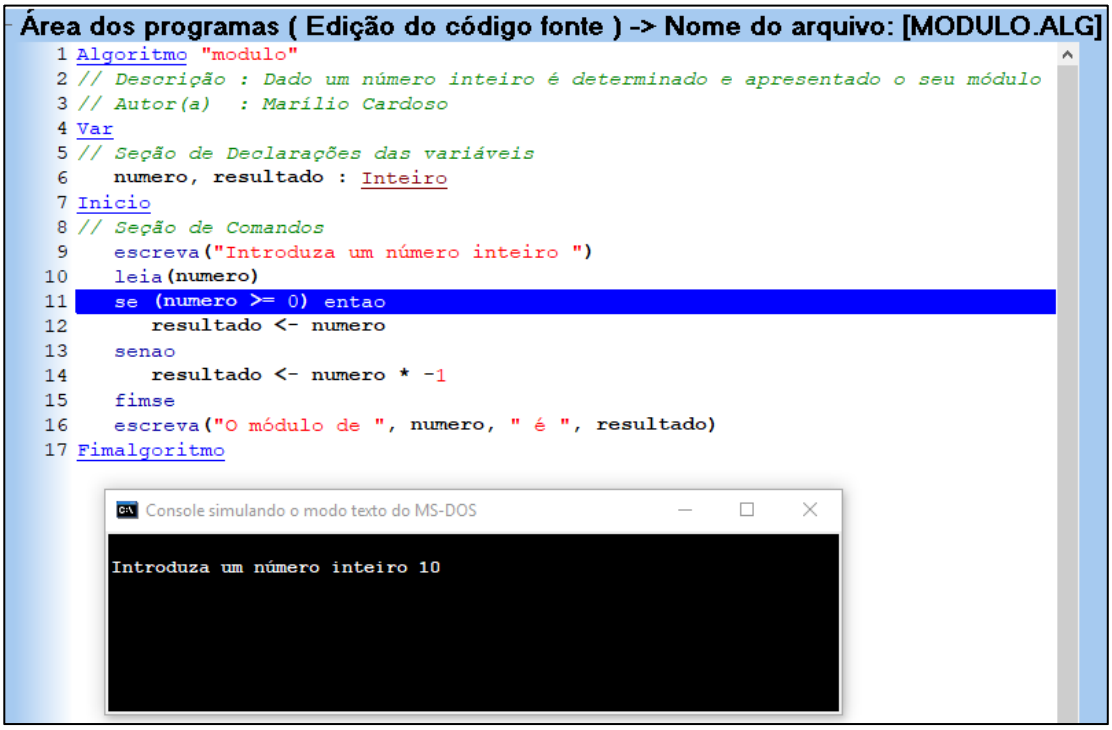


Figure 41 - Step-by-step execution in VisuAlg

It is also observed in Figure 42 that the variable "NUMBER" has the value entered by the user (10) and that the variable "OUTPUT" contains the value 0, with which it was initialized in its definition, because no instruction has been executed yet that would change that value.

Áreas das variáveis de memória (Globais e Locais)

Escopo	Nome	Tipo	Valor
GLOBAL	NUMERO	I	10
GLOBAL	RESULTADO	I	0

Figure 42 - Value of the variables in VisuAlg in a "step by step" execution

In case there is an error such as a variable being invoked without being defined, a message is presented to the user as shown in Figure 43.

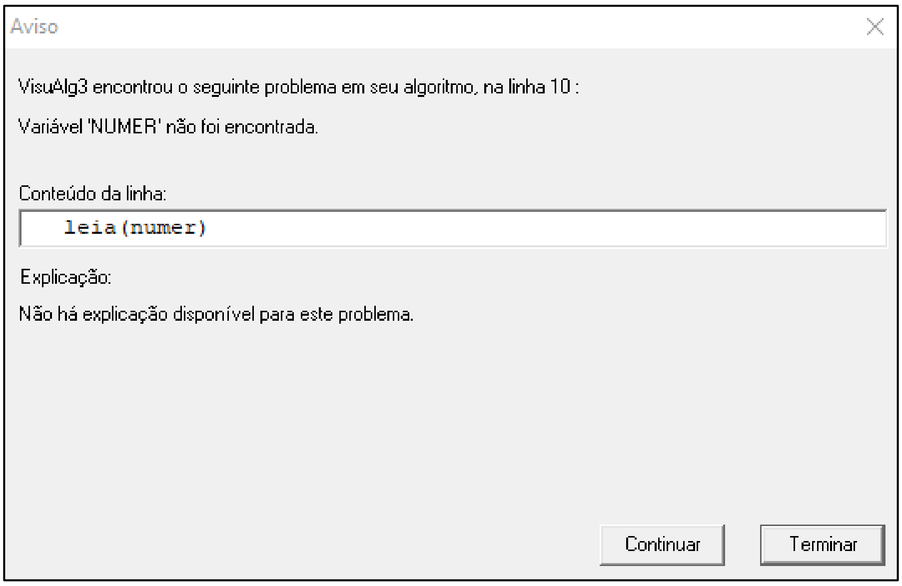


Figure 43 - VisuAlg error message

Also, in the source code the line with the error is marked in red, as shown in Figure 44.

Área dos programas (Edição do código fonte) -> Nome do arquivo: [MODULO.ALG

```
Algoritmo "modulo"
// Descrição : Dado um número inteiro é determinado e apresentado o seu módulo
// Autor(a)   : Marílio Cardoso
Var
// Seção de Declarações das variáveis
   numero, resultado : Inteiro
Inicio
// Seção de Comandos
   escreva("Introduza um número inteiro ")
   leia(numer)
   se (numero >= 0) entao
      resultado <- numero
   senao
      resultado <- numero * -1
   fimse
   escreva("O módulo de ", numero, " é ", resultado)
Fimalgoritmo
```

Figure 44 - Line identification with error in VisuAlg

As it has been shown, VisuAlg presents as its main advantage the fact that it allows the development of the algorithm to be brought closer to a context of using a programming language, allowing it to compile and execute, carrying out functional verification tests. It can thus be considered an ally of the teacher and the student allowing for a faster development. Paradoxically, the fact that it can be compiled requires compliance with syntactic rules, which sometimes proves counterproductive, particularly in students with scant previous knowledge. In these cases, there may be a loss of focus of the essential that is the conception of the solution, due to difficulties caused by syntactic errors.

However, if well applied and its operation properly explained, it appears to be a very useful pedagogical tool.

3.4 FACE-TO-FACE TEACHING AND DISTRIBUTED TEACHING

Education can be seen as a gradual process that promotes positive changes in human life and behavior. It increasingly represents a fundamental dimension in the life of each individual, endowing them with the skills that will enable them to develop their activities throughout life. If traditionally education was very much associated with social and economic status, today, even if this relationship has not been broken, it is much more directed towards personal fulfilment.

Traditionally, teaching is carried out in a classroom, in a specific space for that purpose and with the presence of a teacher. However, over time there have been some changes leading to alternative models. Nowadays, with the advent of the Internet and with the generalization of devices that make it possible to access it, educational content has become available in a distributed way allowing students to access it remotely at anytime and anywhere from one of the aforementioned devices.

This new reality provides new ways and means of transmitting and acquiring knowledge in a distributed and remote way, increasing the teaching and learning possibilities.

3.4.1 Types of learning

Knowledge can be acquired in various forms and ways. When we talk about learning, the concept is usually connoted with teaching, so we are driven to think about school, regular and formal education. However, from the very beginning of each individual's life, learning takes place informally in socialising contexts. Thus, learning can be considered in three dimensions (Figure 45): formal, non-formal and informal (UNESCO, 2018).

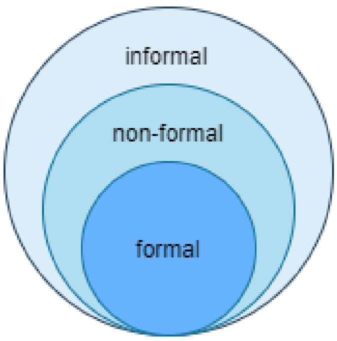


Figure 45 – Types of learning

Although there is a great diversity of understandings regarding the distinction between concepts, although in a simplified way, as established by (Trilla-Bernet, et al., 2014), each of the elements of this trilogy can be associated to specific contexts. Formal learning is strongly connoted with regular education, in an institutional and organized education system, with informal learning associated with actions carried out outside school but sustained by intentional and organized processes. Informal learning usually occurs in contexts that are neither intentional nor structured for the purpose, such as family, friends or the general community (Esclapez, 2008).

The distinction may also depend on factors such as the place where it occurs, the learning process, contents and objectives (Van Noy, James, & Bedley, 2016). There are also those who advocate that the distinction should include other factors such as motivation, interest, social context and evaluation (Eshach, 2007).

Despite the diversity of opinions and lack of consensus on several aspects and the absence of clearly defined boundaries, it is nevertheless possible to identify some specific characteristics of each type, which are presented below.

3.4.1.1 Formal education

It usually refers to an institutional structure, planned and organized, guided by a formal curriculum (Gohn, 2006), supported by public or private, accredited and recognized educational institutions that together constitute the educational system of the country. In this context, certifications and curricula are established and/or recognised by government authorities and education typically generates a diploma of completion or recognition of qualifications. Formal learning includes compulsory education (primary and secondary), but also higher education, adult education and vocational education, typically taking place in classrooms or spaces specifically designed for this purpose.

3.4.1.2 Non-formal learning

In non-formal learning, the educational process typically occurs outside the school, in a more relaxed and less organized context, albeit one that can be guided by a curriculum (Perraton, 2007). This process can be promoted by institutions not necessarily linked to education, such as foundations, political parties, clubs, associations, youth organizations or trade unions, but also by institutions or services created to complement the formal education system such as music classes or exam preparation classes. They are usually flexible contexts, with active and learner-centred approaches.

Although this type of teaching is not usually aimed at a qualification, it can be quite enriching by contributing to the reinforcement of the technical, social and cultural abilities of the individual (Eshach, 2007). It is often considered more motivating as it is less rigid and typically has more interested and participative students. It can be complementary or alternative to formal education and is an important component in lifelong learning.

3.4.1.3 Informal learning

Informal learning results from daily life processes and experiences, without systematisation or organisation. It can be the result of mimicry processes by direct observation of the activity of others, or it can come from various sources, such as the media.

Informal learning can occur in an intentional way, if there is interest and pursuit of knowledge, but also in an accidental or casual way. In such cases, being neither intentional nor, in many cases, conscious, it tends not to be recognised, even by itself, as contributing to the increase of its knowledge and skills.

According to (Trilla-Bernet, et al., 2014) an educational process that does not result from something specific and that occurs in an undifferentiated and contextualized way in other social processes, inseparable from other cultural realities, occurring in an imprecise way and without a clear outline, it will be a case of informal learning.

3.4.2 Face-to-face education

The concept of face-to-face teaching is, empirically, associated with the traditional teaching model, in which learning takes place in a classroom, with the simultaneous physical presence of students and teacher, in a specific place and at a specific time (Lima & Capitão, 2003). This is the model that has been used for several centuries, from ancient Greece, through the Society of Jesus with the application of its Ratio Studiorum (Duminuco, 2000), to the present day, and therefore there is vast experience in its use.

The process is conducted by the teacher, who acts on the basis of indicators he or she obtains from the students' attitude. Typically, the teacher imparts information to all students simultaneously, with less frequent individualized action or action aimed at small groups. While technological means are increasingly used in this type of teaching, the most relevant and important factors are the stance of the teacher, his verbal and non-verbal language and the use of vocal intonation techniques (Moore & Thompson, 1997).

Despite some inconveniences identified in face-to-face education, such as inflexibility of schedules or travel costs to the training site, it does, however, have some characteristics that give it a unique and distinctive character.

These may include direct student-teacher interaction, which allows real-time clarification of doubts, relationships that are established between the students themselves, reinforcing social ties and enhancing future professional networking, promoting aid and solidarity and the development of soft skills, such as teamwork, negotiation, conflict management, decision-making and communication. In addition, the teacher can more easily identify the difficulties experienced by students through their reactions and can redefine strategies or reinforce actions to fill identified shortcomings.

In higher education, the teaching of programming is traditionally organized in classroom classes where the concepts are presented and explained, and classes of a practical and/or laboratory nature where it is intended that students practice applying theoretical knowledge (Castro A. V., 2005). Classes are organized on a weekly basis, with pre-defined contents, usually at the same time and place for each class/course unit. In this rigid model, the occurrence of an event that prevents the class from happening, such as the teacher's absence or holidays, strongly conditions teaching and learning. Also, the absence of one student, having no impact on the others, can be very revealing for the student himself, who must compensate for this absence with independent study or with the help of colleagues.

3.4.3 Distributed education

In an ever more digital and globalized world, technology is present in the most diverse contexts and areas of activities. Also, in education there is an increasing use of the same for the most diverse purposes, whether in the context of the classroom, libraries, media libraries, laboratories or for self-learning using online resources. This comes from social, cultural, professional and technical factors, overcoming difficulties of access to training through face-to-face teaching, fighting isolation and inwardness, and making it possible to increase qualifications. According to Moore (Moore & Kearsley, 2011) distance learning (D-Learning) is both an effect and a cause of structural changes in practices and the concept of education.

In short, D-Learning results in remote learning that is mediated or not by technology, and teachers delivering their teaching resources and activities upstream to which students can access and respond. In the case of e-Learning, these activities can even occur at any time and from any place as long as the minimum conditions exist for this purpose, such as a computer and internet access.

This new scenario, with a myriad of resources and without geographical boundaries, presents new opportunities and challenges to students, teachers and educational institutions.

The term distributed education has been used to refer to D-Learning heavily mediated by technology. Although regarded as relatively modern, the first evidence of D-Learning is almost two centuries old, when in 1840, in England, Sir Isaac Pitman organized a correspondence course (Peres, Mesquita, & Pimenta, 2015), the Correspondence Colleges, to teach his shorthand method. After this first action which used the mail as a medium, and with the technological advances, the use of radio, cinema and television for educational purposes was observed. Today, with the advent of computers, the Internet, various equipment and technologies and other resources, there has been an exponential increase in the possibilities of using the DL, enhancing access to information and training for a greater number of individuals.

A historical landmark of D-Learning in Portugal was the beginning of the school by TV in the mid-60s of the last century, although there had already been previous D-Learning experiences with correspondence courses. There are, however, those who, as a Trindade (Trindade, 1990), consider that the school by TV does not correspond to the concept of DL, since, although supported by audio-visual means, its methodology follows that of face-to-face teaching. In 1980, the Institute of Bank Training (IFB)[67] was created, an organ of the Portuguese Bank Association (APB) to carry out training actions, which uses the face-to-face and distance learning methods (Lagarto J. R., 2002) and in that same year, the Portuguese Institute for Distance Learning was formally created, by Decree-Law no. 519-V1/79[68], as an embryonic organism for the future creation of an Open University, which was created only eight years later, by Decree-Law no. 444/88[69], of 2 December 1988.

In the case of Portugal, the Open University[70] is today a reference institution in D-Learning, having established with the Portuguese State an institutional development contract with a minimum tenure of five years. This contract includes, among other objectives, the specialization in scientific and pedagogical competences and methodologies as well as in infrastructures and D-Learning systems, besides the administration of study cycles exclusively in the D-Learning mode[71].

There are several other examples of higher education institutions based on the D-Learning model in the world. One of the best-known and most paradigmatic is that of Universidad Nacional de Educación a Distancia (UNED)[72] (the largest university in Spain, with more than two hundred and fifty thousand students from fourteen countries and three continents and over a hundred research groups. UNED's motto is "Be where you are" ever since it was founded in 1972 as a D-Learning model, making it possible to obtain an academic degree in a flexible way, at the student's pace, anytime and anywhere.

Another very important institution in this context is The Open University (OU)[73], founded in England on April 23, 1969. According to OU data, its student universe surpasses two hundred and fifty thousand and its area of influence extends to one hundred and fifty-seven countries.

D-Learning has specific characteristics such as:

- physical separation between the teacher and students;

---

[67] https://ifb.pt/cursos-ifb/
[68] https://dre.pt/application/conteudo/157039
[69] https://dre.pt/application/conteudo/357130
[70] https://portal.uab.pt/
[71] https://dre.pt/application/conteudo/124392062
[72] https://www.uned.es/universidad/inicio.html
[73] http://www.open.ac.uk/

- the student as the central element of the process;
- temporal and spatial flexibility in learning;
- more autonomous learning at the pace chosen by each student;
- stakeholders need to be equipped with digital competences;
- use of synchronous and asynchronous communication tools;
- need for initial investment.

Its particularities result in various benefits for learning, such as studying from any place and at the time that is most convenient for the student, thus developing their autonomy skills.

In this way, the geographical or temporal constraints are overcome, allowing work and study to be compatible, particularly in cases of shift work or frequent travelling, of those displaced on missions, or even of individuals in geographical areas without training offered by higher education institutions.

Nevertheless, this flexibility is a challenge, as it makes the student more responsible, implying that he/she is motivated and focused, since there is no commitment to attend a class at a predetermined time or to present him/herself before the teacher. In younger audiences, with less maturity, less responsibility or lack of personal organisation, this can represent a problem, strongly conditioning the chances of success. Thus, D-Learning cannot be seen as an exclusive model suitable for all situations and realities, with face-to-face teaching being the most appropriate (possibly indispensable) for teaching certain contents and/or specific populations.

3.4.3.1 E-Learning

In this digital age the prefix ″e-″ is extremely common and can be found in words such as e-mail, e-business or e-book. Also, in education the term e-Learning (electronic learning) is widely used, even though the concept is not consensual, with various meanings associated with it and various forms of application in an educational context.

Although D-Learning and e-Learning are often used as synonyms, e-Learning is not only a form of D-Learning (Rosenberg, 2006), but rather substantially increases the possibilities of D-Learning (Monteiro, Moreira, & Lencastre, 2015). However, in general terms, e-Learning could be mentioned as a form of non-presential teaching designation, with organised teaching resources, and with teaching mediation, supported by ICT.

In this context, the role of the teacher differs substantially from face-to-face teaching, beginning with the need for the teacher to have technical skills in the use of digital resources (Miranda, 2009). It is also up to them to define and prepare the contents in an articulated way with the resources that they or others can make available to their students. The role of the teacher is no longer one of transmitter of knowledge, but rather one of facilitator and mediator of learning.

In addition, there is a need for communication with the student which can occur synchronously or asynchronously. This requires resources and tools such as *e-mail*, discussion forums, chat, frequently asked questions (FAQ) or video conferencing.

From the student's perspective, e-Learning can be seen as a process of access to digital content, enabling autonomy and self-learning, supported by an instructor who assists them in carrying out their activities and clarifying doubts. Thus, learning can be personalised and appropriate to the student's pace, needs and availability, with the student being responsible for managing their own learning path.

Besides students and teachers, e-Learning implies other actors such as technicians, designers and administrators, with technical and management responsibilities necessary for the development of activities.

This type of training presents clear advantages, responding to the needs felt by society in the search for knowledge and qualification (Peres, Mesquita, & Pimenta, 2015). However, it has some particularities that can be regarded as less positive and/or potentially problematic. In addition to the above-mentioned ones concerning immature audiences, communication difficulties may arise, as possible doubts raised by the student may not be answered by the teacher in due time. Also, with regard to technology, it can be seen as an obstacle by some students apart from depending on devices and technical solutions which, however reliable they may be, always present some probability of failure or unsatisfactory operation. For example, obsolete or low internet speeds (Peres, Mesquita, & Pimenta, 2015).

3.4.3.2 B-Learning

As can be seen from the above, face-to-face teaching and D-Learning are not in competition, but complementary. By combining these two models it is possible to mitigate the drawbacks of each model, thus resulting in a mixture called blended Learning (b-Learning) or mixed teaching (Peres & Pimenta, 2011). In b-Learning teaching is based on digital content with *online access*, and usually supported by an LMS, with moments of autonomous learning coexisting in a virtual way, with face-to-face sessions fostering student-teacher and student-student interaction (Marques, 2011).

It can be considered that this is the model adopted at ISEP, in which, in addition to the face-to-face classes, there are contents and digital tools, available remotely, which allow students access to information and the possibility of carrying out tasks to consolidate learning. These two aspects (face-to-face and online) are complementary, enabling personalised and more attractive learning.

According to Marques (Marques, 2011) there are several reasons for using b-Learning which can, however, be summarised in three main groups: benefits for the pedagogical process, improvements in logistics and in access to content and also economic and financial advantages (Figure 46).

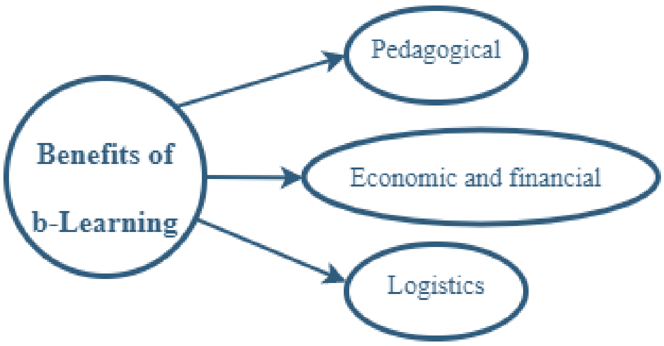


Figure 46 – Benefits of b-Learning

In the pedagogical field, the diversity of methodologies, approaches and resources make it possible to meet the differentiated profiles of students and the various learning styles (Alammary, Sheard, & Carbone, 2014). B-Learning has proven to be appropriate to the characteristics and needs of a very heterogeneous universe of students of different ages, motivations and educational and socio-cultural backgrounds (Monteiro, Moreira, & Lencastre, 2015).

B-Learning involves a logistics that is suited to the aforementioned diversity of student profiles, as well as to the existence of many student workers and professionals in need of updating and/or retraining. Flexibility is one of its important features enabling access to education for those who have geographical, time or other restrictions to attend classes in person on a regular basis.

In the economic side, b-Learning promotes a reduction in costs when compared to face-to-face education, particularly as regards travel and the use of infrastructure. It also represents an added value in terms of business opportunities for institutions by providing a diversified training offer and reaching new and/or geographically dispersed audiences. Sometimes b-Learning is adopted by institutions as a way of starting teaching online (Littlejohn & Pegler, 2007).

3.4.3.3 M-Learning

The "democratization" of access to communications and mobile devices and the proliferation of *wireless* networks, associated with the development of numerous mobile applications in recent years, has boosted the massive use of these means for the most varied purposes. Education also proved to be an interesting application context giving rise, in 2005, to the concept of mobile learning or m-Learning (Crompton , 2013).

The m-Learning can be understood as a set of educational processes supported by ICT and mobile devices, where students are in different places and geographically dispersed, which may be formal learning spaces or any other (Saccol, Schlemmer, & Barbosa, 2011). The constant technological evolution and the diversity of technical solutions in terms of equipment make it difficult to precisely define what is considered a mobile device. UNESCO uses a broad definition, considering as mobile devices digital equipment, portable, with Internet access, capable of performing a set of tasks, particularly with regard to communication and, typically, controlled by an individual and not by an organization (UNESCO, 2013).

The use of devices with which students are already familiar and which they use daily as mobile phones, smartphones, smartwatches, computers and tablets, combined with the ease of access to the Internet anytime and anywhere, are all aspects that facilitate and motivate their use for educational purposes (Shih & Mills, 2007).

The use of mobile technologies, in isolation or combined with other means, is an enabler of diverse learning, allowing not only autonomous learning but also collaborative learning (UNESCO, 2013). However, this context presents technological but also pedagogical requirements in that curricula and teaching methodologies must be adapted to this reality (Churchill, Jie, Chiu, & Fox, 2016).

M-Learning establishes a link between formal and non-formal learning by providing the student with access to information that can clarify or complement concepts taught by the teacher. Today there are countless digital resources from videos to mobile applications with detailed, step-by-step explanations of the most varied subjects, allowing students to learn autonomously and at their own pace.

Another interesting aspect of the use of electronic devices is the possibility of very fast or even immediate feedback, allowing problems to be identified without resorting to the teacher, not limiting the pace of student study and increasing motivation. For this purpose there are applications such as tests, challenges, quizzes, games among others, which allow for a more playful and attractive learning, as well as obtaining information on the progress of learning, thus being important allies of the student, but also of the teacher, allowing a faster, earlier and more accurate diagnosis of the evolution of the students (Al-Emran, ElSherif, & Shaalan, 2016).

3.4.3.4 U-Learning

The concept of ubiquitous learning (u-Learning) (Mesquita, Moreira, & Peres, 2016) is associated with learning processes supported by digital technologies, either mobile or otherwise, which establish a link between the educational context and the social and physical environment and the daily life of the student, making it possible to bring the face-to-face and virtual environments closer together. This mix between virtual and real, linking people, places, objects, activities and contents enriches the learning context and enhances continuous and meaningful learning (Saccol, Schlemmer, & Barbosa, 2011). For Huang et al. (Huang, Chiu, Liu, & Chen, 2011) u-Learning is a learning modality derived and extended from e-Learning and m-Learning in which students can learn in a profound and ubiquitous way. Other authors refer to u-Learning as everyday learning, supported by technology and combining physical spaces and virtual environments, with activities taking place naturally without restrictions of time or space (Tu, McIsaac, Sujo-Montes, & Armfield, 2016).

3.5 LMS

The term Learning Management System (LMS) generally refers to an online system that allows the management of courses online, making content available and promoting communication (predominantly asynchronous) and cooperation between students and teachers. It consists of a Web platform for managing teaching-learning processes in the technical, administrative and pedagogical dimensions (Peres & Pimenta, 2011).

3.5.1 Introduction

The provision of resources by the teacher is an established practice since the beginning of teaching, with the books to be consulted and/or provided notes or other physical resources to support the study being indicated by the teacher.

With the widespread use of computers and, in particular, with the emergence of the Internet, digital educational resources have grown exponentially, radically changing access to information and the way of studying and researching.

The first approach to a mechanism for sharing digital resources was to use each teacher's personal pages, from which students could remotely access the content made available, when they so wished.

As an example, the page of a DEI/ISEP teacher[74] is shown in Figure 47.

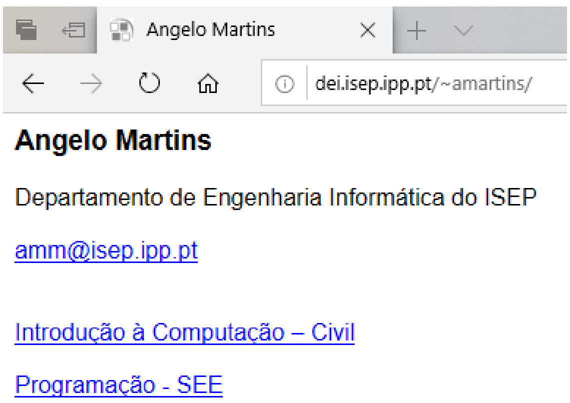


Figure 47 - Personal page of a teacher

[74] http://dei.isep.ipp.pt/~amartins/

As can be seen, the teacher made his personal contact available to provide students with a digital means of communication and two subjects: Introduction to Computing, in the course of Civil Engineering and Programming, in the course of Electrical Engineering - Electrical Energy Systems. By selecting the first link the one shown in Figure 48 is obtained.

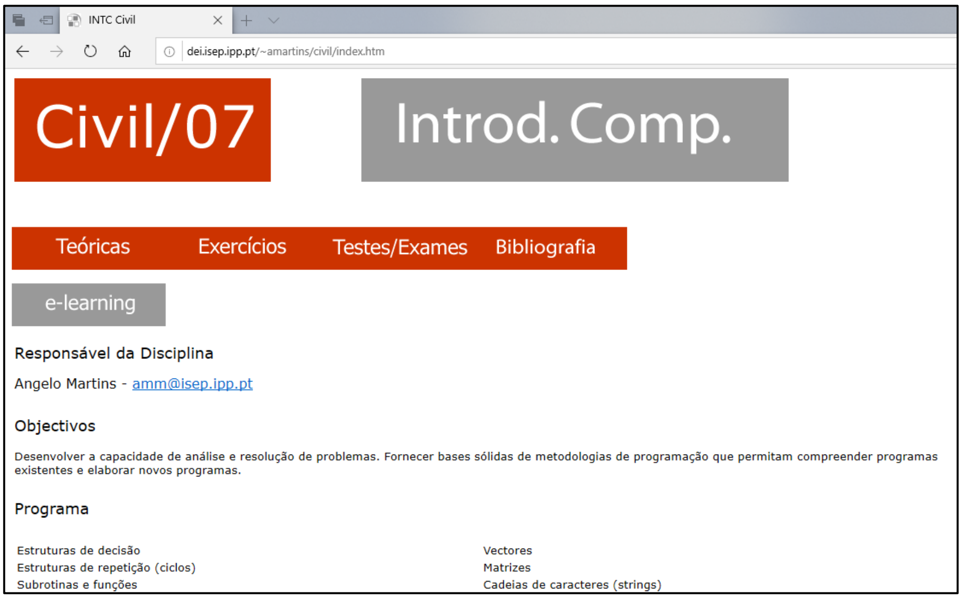


Figure 48 - Page of a curricular unit in the 2006/2007 school year

As can be seen, the teacher has chosen to organise his pedagogical contents in four sections: Theoretical, Exercises, Tests/Exams and Support Bibliography, corresponding to links to other pages. As an example, we can see in Figure 49 part of the contents of the theoretical lessons page.

Figure 49 - Introduction to Computing lectures page - Civil Engineering

On this page we can find links to documents that can be viewed online by the student or, in some cases, downloaded for subsequent viewing.

As such, this is a means for the student to access content and obtain information, and the teacher can thus establish unidirectional and asynchronous communication with the students.

Technological evolution in terms of access, bandwidth and internet speed, the appearance of LMS and its uptake as an institutional tool in several organizations, has made this practice of using personal pages fall into disuse. In addition to the increase in the speed of access to content via Web, the development of compression techniques (codecs) has boosted the production of multimedia teaching content and, in parallel, the increase in its use (Castro A. V., 2012).

The LMS allowed for the integration of functionalities of several digital tools that operated separately. Having emerged as tools to support e-Learning (Carvalho A. A., 2008), they quickly began to be used in various contexts, particularly as an indispensable tool for b-Learning.

The use of an LMS should make it possible to simplify training management, in particular off-site training, by assisting trainers and trainees. Some of its most common features are the management of courses, in the most varied aspects, from user registration (teachers and students), through the storage and management of content, to the performance of evaluations and management of scores. It also enables communication between the stakeholders, in a synchronous and/or asynchronous manner, as well as access, activity and time recording in the system (Lagarto & Andrade, 2009).

An LMS must meet some specific requirements in order to fulfil its mission flexibly, effectively and efficiently. These include high availability, scalability, stability and security, as well as ensuring interoperability between systems and resources from different sources, based on open standards for deployments on the Web.

An LMS typically consists of two main parts:

- a server where course and user creation and management operations, authentication checks and user notifications are performed;
- a graphical user interface (GUI - Graphical User Interface).

The server is usually located in the institution and is accessible to its users via Internet from anywhere.

LMS provide means of sharing educational resources, synchronous and asynchronous communication, such as chat rooms, video conferences and discussion forums, in addition to increasingly resources for educational purposes such as questionnaires, referenda and tests (Castro A. V., 2012).

3.5.2 Examples of LMS

There are several LMS, many of them developed and marketed by companies while others are free and open source. Free and open source software enhances its development and dissemination by allowing the development of new resources, the improvement of existing ones, the correction of deficiencies and the sharing of information about problems that have occurred and how to solve them.

Some of the best known and used LMS in the context of education are: Blackboard, Moodle and Sakai.

3.5.2.1 Blackboard

Blackboard is a commercial product, created by the software company of the same name[75] based in Washington and founded in 1997. As of 2005 the company merged with WebCT, a pioneering company in the development of learning platforms, at that time used by about 2200 institutions in over 60 countries (Zazpe, 2017).

It is a widely used LMS in North America (Martin, 2008) and was developed with the aim of providing an online potentially enriching learning environment. In Portugal it is used in some higher education institutions, two of the most relevant being the University of Minho[76] (Carvalho, Areal, & Silva, 2011) and ISCTE - Instituto Universitário de Lisboa[77].

It is a virtual environment dedicated to D-Learning with communication tools primarily asynchronous (Castro A. V., 2012), as well as several other functionalities such as content management and sharing, evaluation or interconnection with other systems. Today it represents more than just an LMS, corresponding to a family of tools, the most representative being Blackboard Learn.

3.5.2.2 Sakai

Sakai[78] originated from a collaborative project for the development of a free, open source software platform, developed in Java and distributed under the license of the educational community, which resulted in an online collaborative teaching-learning tool.

The founding entities of the project were the Universities of Michigan and Indiana in the United States of America, which were later joined by the also American Stanford University and MIT, in addition to the uPortal[79] and the Open Knowledge Initiative (OKI). The project also had financial support from the Andrew M. Mellon Foundation[80] (Zazpe, 2017). To manage Sakai, a foundation was created that brought together more than a hundred universities and other organizations.

Sakai's features are organized in four tool categories (Zazpe, 2017): collaborative, teaching and learning, administrative and portfolio. Perhaps the most innovative and differentiating feature from other LMS is the e-portfolio. The focus of Sakai is on collaborative work, and the e-portory management system it has serves that purpose, allowing stakeholders to publish and share their work and consult and view the portfolio of other users (Castro A. V., 2012).

3.5.2.3 Moodle

Moodle[81] is a result of the acronym Modular Object-Oriented Dynamic Learning Environment. However, the word moodle is also a verb associated with unpretentious navigation and concomitant with other activities in a pleasant, relaxed and creative environment (Castro A. V., 2012).

At ISEP, the teaching model is rooted in Moodle and is widely used in all courses and for various purposes. Its regular use dates back to the 2006/2007 school year, when it was adopted as a form of systematization of procedures and pedagogical and technical support to students and teachers, aiming at making the teaching-learning process more flexible and improving it.

This is an open source platform for managing the teaching-learning process, developed by Martin Dougiamas at the beginning of this century and created with the purpose of supporting

[75] https://www.blackboard.com/
[76] https://elearning.uminho.pt/
[77] https://e-learning.iscte-iul.pt/webapps/login/?new_loc=%2F%40%40
[78] https://www.sakailms.org/
[79] https://www.apereo.org/projects/uportal
[80] https://mellon.org/
[81] https://moodle.org/

a constructivist approach to teaching (Dougiamas & Taylor, 2003). Moodle is one of the most widely used LMS in the world (Lopes, 2011) (Rodrigues, Rocha, & Abreu, 2017), with about 20 million courses and 175 million users[82].

It consists of a modular platform that allows the incorporation of new modules and tools, and provides a diversity of features that enable the teacher to meet the needs of students (Cole & Foster, 2007). It allows, for example, the creation of courses or course units, assign users and define their profile/role.

In Moodle it is possible to assign several basic user profiles[83], as shown in Table 10.

Table 10 - Moodle user profiles

Profile	Function
Administrator	Has permissions to perform any type of action on all resources
Manager	Has management functions, but with permission for a smaller number of actions and/or with some restrictions
Course Creator	Can create and manage courses
Teacher	Can manage courses and add content
Non-Editor Teacher	Can access courses, but with no editing permissions
Student	Can access and participate in courses
Guest	Can view courses, but cannot attend these
Authenticated Users	Role that all users have
User authenticated in first page function	User authenticated for first page function only

New profiles with differentiated permissions can also be added, and it is advisable to create new profiles based on existing ones instead of changing them.

Within Moodle it is possible to define course and course unit managers, control and monitor access, and create and manage discussion forums. It also allows for the inclusion of assessment systems based on the potential for automation of learning systems, such as multiple-choice tests or gap-filling tests, with questions generated randomly from a database.

Being a free and open source platform, it is constantly evolving, and is supported by an active and numerous community of programmers. There are currently many plugins from Moodle (over 1500) (Moodle, 2019) for a multitude of features, making it robust and versatile as it is constantly being evaluated and improved and allows for specific configurations tailored to the needs and interests of each organization (Al-Ajlan & Zedan, 2008).

As an LMS, Moodle also works as a Learning Content Management System (LCMS), since it allows content management enabling its creation and editing. Its basic functionalities and the tools it supports present a diverse set of solutions that cover the overwhelming majority of the pedagogical needs of the institutions that use it. Among the available features are: chat, discussion forums, glossary, tasks, calendar, quizzes and wiki.

[82] https://stats.moodle.org/
[83] https://docs.moodle.org/37/en/Standard_roles

4 SUPPORT MECHANISMS FOR PROGRAMMING TEACHING

"You cannot create experience. You must undergo it."

Albert Camus

This chapter aims to present the concept of a programming teaching support mechanism, in particular with regard to automatic code evaluation, describing the functionalities of the tools used for this purpose.

Several tools are also identified, and for some of them a summary description of their origins, functionalities and applications is made.

At the end of the chapter the reasons for choosing the VPL as the tool to be used in APROG are presented, based on criteria defined for this purpose.

4.1 INTRODUCTION

Learning to program involves an iterative process, but there are differing opinions on how to do it most effectively. Some people like Tavares (Tavares, Henriques, & Gomes, 2015) argue that you can only learn to program by programming. Gomes (Gomes A. , 2010), however, considers that "one learns to program not only by programming, but by studying and reflecting on the way one has programmed". In either case and regardless of the author, training is considered as one of the most important steps in the learning process of programming (Cheang, Kurnia, Lim, & Oon, 2003), and practice with regular repetition of the resolution of a large number of exercises to consolidate learning is crucial. There is, nonetheless, an essential aspect to training which is feedback, which should be fast in order to enable the student to locate errors, understand them and correct them, avoiding demotivation (Pelz, Jesus, & Raabe, 2012).

Errors in programming can be divided into three groups: syntax, execution or logic.

Syntax errors occur when the programmer does not obey some grammar rule of the programming language used, causing the interpreter or compiler to be unable to translate the source code. They are related to the structure of the program and the writing rules of the code. Currently any IDE provides code writing aids, minimizing syntax errors. Still, if the code contains errors, the compiler will detect them and the programmer will receive that information, allowing them to identify and correct the errors.

There may also be runtime errors, resulting from poor code design, which are not detectable in the compilation, causing the program to be stopped abruptly or, in some cases, entering an infinite cycle and not finishing execution. Some common errors of this type are the attempt to access unavailable or non-existent external devices, the division by zero, the invocation of a non-existent memory reference, or the assignment of values to variables of incompatible types.

Logic errors are usually less explicit and more difficult to detect, not being identified by the interpreter or compiler, so the program runs but produces erroneous results. This error type may result from misinterpretation of the specifications, logical-mathematical errors, or from the programmer's mistake in coding. Typical errors of logic are not initializing variables or initializing variables with incorrect values and checking the state of variables at inappropriate places in the code. These errors can be detected by using debuggers that allow the program to run instruction by instruction and, at any given moment, to know the status of each variable. There are also other solutions to avoid or minimize logic errors, among which is the use of unit tests which allow testing individually each block of code that performs a specific task.

In programming training several errors of the above types are likely to occur. However, if in the case of syntax or execution errors, they are obvious to the programmer, logic errors may not be immediately detected.

If exercises are carried out without supervision, for example from the teacher, and without any feedback, those who have implemented them may not know whether the results obtained were in fact those expected. In addition, even if the programme is effective, achieving the expected results, it may not be well structured or function efficiently, so supervision and advice is crucial.

Rating and providing significant feedback is a popular and effective method for involving students in programming (Verdú, et al., 2012), with the lack of feedback and monitoring by the teacher of student work being one of the most important difficulties associated with the learning process of programming (Koulouri, Lauria, & Macredie, 2015).

Code analysis and evaluation is usually a time-consuming, error-prone and tedious task, with each program being tested and its source code analysed. Moreover, due to tiredness, distraction, favouring or any other factor, the human analysis is subjective and different evaluators may assign different classifications to the same work (Fonte, da Cruz, Gançarski, & Henriques, 2013) or even the same evaluator may classify similar resolutions differently if analysed at different times.

These factors condition the amount of exercises to be solved by the students and the obtaining of quick feedback (Ala-Mutka, 2005). The automatization of the code evaluation process, with electronic submissions and immediate feedback is a means to increase the volume and pace of training.

Immediate feedback should act as a motivating element for the student as it provides them with satisfaction in obtaining results from their work, usually in an evolving process of increasing difficulty. However, in the case of frequent or repetitive failure, feedback can lead to anxiety, frustration and demotivation, so it is advisable that systems can mitigate this effect by, for example, inhibiting submissions for a period of time after several failures. The teacher should also be aware of this by regularly checking submissions and results so that they can intervene in good time.

4.2 AUTOMATIC EVALUATION MECHANISMS

Contrary to what one might think, automatic program validation mechanisms are not a recent issue. As early as 1960 an article was published in the Association for Computing Machinery (ACM) magazine (Hollingsworth, 1960) on automatic evaluation in programming.

With the development of computers, systems and applications, with the appearance of more programming languages and applications, there has also been a profusion of systems dedicated to automatic code evaluation. The operation of these systems normally includes the receiving of the code sent by the user, its compilation, function tests and classification.

There are systems in place for various purposes, such as for programming competitions, for educational purposes in teaching programming or for selecting and recruiting programmers. Although some of these systems operate only locally, several have been designed for the Internet or to be integrated into existing systems or LMS. Some of these systems are quite versatile and can be used in more than one of the contexts mentioned and allow the use of several programming languages (Wasik, Antczak, Laskowski, & Sternal, 2018) (Gupta & Gupta, 2018).

For this work, a bibliographic search was performed in the ACM, Elsevier (ScienceDirect), ERIC (EBSCO), IEEExplore, Springerlink and Web of Science databases and the Google Scholar search engine. Research was also conducted in scientific repositories, namely in RCAAP[84] (Open Access Scientific Repositories of Portugal) and in Scientific Repository of the Polytechnic Institute of Porto[85] (RECIPP). The survey was conducted in English and Portuguese in the second half of 2016, covering a period of 12 years (since 2005). However, in some cases older publications were selected because they are relevant to the subject and contribute to the establishment of a chronological line of technological evolution. In the research, the following terms were used together: automatic classification, automatic evaluation, programming tasks, learning programming languages and teaching programming. Later, throughout the development of the work, more recent publications were identified, some of which were consulted and used for this work.

When this study began, despite our being aware of the existence of systems for code evaluation, we could not imagine the enormous number of existing and under development systems. Thus, there are many documented systems (Ihantola, Ahoniemi, Karavirta, & Seppälä, 2010) (Caiza & Álamo, 2013) (Christian & Trivedi, 2016) (Wasik, Antczak, Laskowski, & Sternal, 2018), some with many years already and widely tested while others, more recent, still somewhat unstable. There are several open source systems, free and available for use and other proprietary, often developed at universities and for their exclusive use.

As examples of these systems we can mention:

- Assyst (Jackson & Usher, 1997);
- AutoLEP (Tiantian, Xiaohong, Peijun, Yuying, & Kuanquan, 2009);
- Automata (Srikant & Aggarwal, 2014);
- BOCA (Campos & Ferreira, 2004);
- BOCA-LAB (arising from BOCA) (Nazário & Souza, 2010);
- BOSS/BOSS2 (Joy, Griffiths, & Boyatt, 2005)
- Bottlenose (Sherman, Bassil, Lipman, Tuck, & Martin, 2013);
- Code Analyzer for Pascal (CAP) (Schorsch, 1995);
- Codeboard[86];
- CodeRunner (Richard & Harlow, 2016);
- CourseMarker (Higgins, Hegazy, Symeonidis, & Tsintsifas, 2003);
- Code Submission Evaluation System[87] (CSES);
- EduJudge (Wasik, Antczak, Laskowski, & Sternal, 2018);
- eGrader (Al Shamsi & Elnagar, 2012);
- Environment for Learning to Program (ELP) (Truong, Bancroft, & Roe, 2003);
- Generic Automated Marking Environment (GAME) (Blumenstein, Green, Nguyen, & Muthukkumarasamy, 2004);
- Homework Generation and Grading (HoGG) project (Morris, 2003);
- IT VBE (Skūpas, et al., 2013);
- JACK (Striewe, Balz, & Goedicke, 2009);
- JPLAS Java Programming Learning Assistant System (Funabiki, Matsushima, Nakanishi, Watanabe, & Amano, 2013);
- Marmoset (Spacco, et al., 2006);

[84] https://www.rcaap.pt/
[85] https://recipp.ipp.pt/
[86] https://codeboard.io/
[87] https://cses.fi/

- Mooshak (Leal & Silva, 2003);
- Programming Contest Control (PC2) *system* (Clifton, 2010);
- PETCHA (Queirós & Leal, 2012);
- Pythia (Combéfis & de Saint-Marcq, 2012);
- Qualified[88];
- RoboLIFT (based on Web-CAT) (Allevato & Edwards, 2012),
- RoboProf (Daly, 1999),
- System for Automated Assistance in Correction of Programming Exercises (SAC), (Auffarth, López-Sánchez, i Miralles, & Puig, 2008);
- VPL (Rodríguez-del-Pino, Rubio-Royo, & Hernández-Figueroa, 2010);
- Web-CAT (Edwards & Pérez-Quiñones, 2008);
- YAP3 + APAC (Pohuba, Dulík, & Janků, 2014).

Table 11 gives a summary of these tools and some of their features, in particular about the programming languages supported, the mode of operation and their main purpose.

[88] https://www.qualified.io

Table 11 – Automatic code evaluation systems

Tool	Supported Languages	Functioning	Purpose
Assyst	C, Ada	Standalone	Teaching
AutoLEP	C	Standalone	Teaching
Automata	C	Web	Teaching
BOCA	Several	Web	Competition, teaching
BOCA-LAB	Several	Plugin	Teaching
BOSS/BOOS2	C, Pascal, Java	Web	Teaching
Bottlenose	Several	Web	Teaching
CAP	Pascal	Standalone	Teaching
Codeboard	Several	Plugin	Teaching
CodeRunner	Python, C, C++, Java, PHP, JavaScript Octave and MATLAB	Moodle plugin	Teaching
CourseMarker	Java, C++	Standalone	Teaching
CSES	Pascal, C++, Java., Python	Web	Competition
EduJudge	C, C++, Pascal, Java	Web	Competition, teaching
eGrader	Java	Graphic	Teaching
ELP	Java. C	Web	Teaching
GAME	Java, C, C++	Web	Teaching
HoGG	Java	Standalone	Teaching
IT VBE	Pascal and C++	Plugin	Teaching
JACK	Java	Plugin	Teaching
JPLAS	Java	Web	Teaching
Marmoset	Several	Standalone	Teaching
Mooshak	Several	Web	Competition, teaching
PC2	Java, C++, Python and other	Web	Competition
PETCHA	Those supported by Eclipse and Visual Studio	Web	Teaching
Pythia	Python and other	Web	Teaching
Qualified	Java, PHP, Ruby, JavaScript	Web	Recruitment
RoboLIFT	Java	Standalone	Teaching
RoboProf	C++	Web	Teaching
SAC	Java	Web	Teaching
SPOJ	Several (about 45)	Web	Competition
URI online judge	Java, C, C++, C#, Ruby, Python and other	Web	Competition
UVa Edujudge	Pascal, C, C++ and Java	Moodle	Teaching
VPL	Java, C, C++, C#, Ruby, Python and other	Moodle plugin	Teaching
Web-CAT	Java, C++, Pascal	Web	Teaching
YAP3 + APAC	Java, C, C++, PHP and other	Moodle	Teaching

It should be noted that the systems mentioned are very different from each other and have different purposes, characteristics, mode of operation and objectives.

Given the huge number and diversity of tools identified and briefly analysed, some have been selected for further analysis. The scope of this work being teaching, however, some tools designed for competition were analysed, but applied to teaching, which are presented below.

4.2.1 BOCA

BOCA Online Contest Administrator[89] (BOCA) (Campos & Ferreira, 2004) is a system developed by Professor Cássio Polpo de Campos, with the purpose of supporting programming competitions, and is used in programming marathons promoted by the Brazilian Computer Society. These marathons allowed registrations, exclusively, to higher education students.

It is a Web system, developed in PHP and with a PostgreSQL[90] database, using Secure Sockets Layer (SSL), which gives it robustness and security, being easy to install and maintain (Cardoso J. d., 2017).

As in other systems, an important aspect of the process is compliance with the specifications about input and output formats. There are also restrictions on the use of graphical resources, and the name of the files to be submitted must be in accordance with the specifications.

In Figure 50 a view of the BOCA interface is shown.

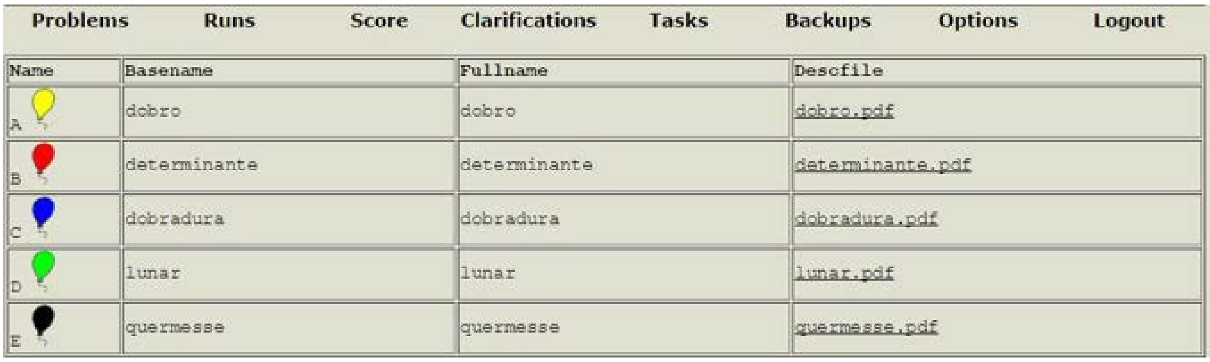


Figure 50 – BOCA Menu[91]

An interesting aspect of this system is that it tests the efficiency of the solution presented, with less efficient resolutions being penalised. However, it has a major drawback in that the evaluation is not automatic, requiring the intervention of a judge whose role is normally played by a teacher.

BOCA, being oriented to contests, presents only the result of the execution, signalling whether the program works correctly or not, but, if it has errors, it does not identify them explicitly. In Table 12 the possible system messages are shown.

[89] https://www.ime.usp.br/~cassio/boca/
[90] https://www.postgresql.org/
[91] https://sites.google.com/site/maratonaunasp/boca

Table 12 – Possible BOCA messages [92]

Response	Description
YES	The program was accepted
NO: Incorrect Output	Also known as *Wrong Answer* It indicates that the program **did not respond to some test(s) correctly**
NO: Time-limit Exceeded	The execution of your program **has exceeded the allowed time**
NO: Runtime Error	A **runtime Error** has occurred during the test
NO: Compilation Error	Program with **syntax errors**. It can also be the name of the problem or language at the time of submission
NO: Output Format Error	Also known as *Presentation Error* Indicates that the program output **does not follow the required** specification, although the "result" is correct
NO: Contact Staff	**Staff presence** required, as an unusual error occurred

As BOCA is an open source system, it quickly evolved into a solution that can now be used in programming laboratories integrated into virtual learning environments. Thus, in this use within the school context, it began to be used as a support to curricular units by allowing students to make submissions and obtaining correction of work.

From this use for educational purposes and its evolution to a plugin, BOCA-LAB (França & Soares, 2012), developed at the Department of Teleinformatics Engineering (DETI) of the Federal University of Ceará (UFC), emerged. As this tool is oriented towards education, it presents some differences from BOCA, of which the elimination of the role of judge and the incorporation of aids stand out, with the student now obtaining feedback which guides them towards the solution. A plagiarism detection feature was also added by using Sherlock (Maciel, Soares, França, & Gomes, 2012).

4.2.2 EduJudge

EduJudge is an e-Learning system developed within the project "Integrating Online Judge into effective e-learning", funded by the European Commission in 2007[93]. The funding was provided in the context of the Lifelong Learning Programme, an axis aimed at improving skills in mathematics, science and technology at European level. To this end, it was intended to stimulate and promote the holding of international programming competitions in secondary and higher education.

The project was based on the UVa Online Judge[94] programming learning tool, one of the oldest and most widely used in the world (Wasik, Antczak, Laskowski, & Sternal, 2018), which was developed at the University of Valladolid in 1995, initially written and developed by the computer student, Ciriaco García de Celis and mathematician and professor Miguel Ángel Revilla[95] (Revilla, Manzoor, & Liu, 2008). Its objective was to make the UVa system and the repository of already existing programming problems available for general pedagogical use in secondary and higher education, updating and standardizing the vast set of already existing exercises and making the tool evolve into a more appealing learning environment, on an asynchronously functioning learning platform. The management and coordination of the project was ensured by the non-profit technology centre, Centre for Telecommunication

[92] https://sites.google.com/site/maratonaunasp/boca
[93] ref. 135221-LLP-1-2007-1-ES-KA3-KA3MP
[94] https://uva.onlinejudge.org/
[95] http://www.eduvalab.uva.es/en/projects/edujudge-project

Development in Castilla y León (CEDETEL), with the participation of the University of Valladolid, University of Porto, Portugal, the Royal Institute of Technology KTH of Stockholm, Sweden and the Institute of Mathematics and Informatics of Vilnius, Lithuania (Revilla, Manzoor, & Liu, 2008).

UVa Online Judge started to be used in April 1997 initially to support C language only. In 1999 and 2000 it supported SWERC (Revilla, Manzoor, & Liu, 2008) and in 2007 it was totally remodelled by professor Miguel Ángel Revilla giving rise to a new version. The educational environment is based on Moodle, on the QUESTOURnament plugin and on the crimsonHex repository, making it possible for the teacher to set up work environments where challenges are set to be overcome by students at a predefined time, in a competitive but also collaborative learning environment.

As mentioned, UVa Online Judge is a system widely used in the context of programming contests (Castro, Perez, Regueras, & Verdú, 2010), with more than 22 million submissions[96] since its creation, as can be seen in Figure 51.

	Total	ANSI C		JAVA		C++		PASCAL		C++11		PYTH3	
1997	4031	3390	84.10%	0	0.00%	639	15.85%	2	0.05%	0	0.00%	0	0.00%
1998	42375	24697	58.28%	0	0.00%	13859	32.71%	3819	9.01%	0	0.00%	0	0.00%
1999	109202	44860	41.08%	0	0.00%	42226	38.67%	22116	20.25%	0	0.00%	0	0.00%
2000	199523	71895	36.03%	0	0.00%	95173	47.70%	32455	16.27%	0	0.00%	0	0.00%
2001	345305	124223	35.97%	3029	0.88%	187852	54.40%	30201	8.75%	0	0.00%	0	0.00%
2002	610151	227779	37.33%	13567	2.22%	305969	50.15%	62836	10.30%	0	0.00%	0	0.00%
2003	874762	282206	32.26%	37726	4.31%	481173	55.01%	73657	8.42%	0	0.00%	0	0.00%
2004	998194	279821	28.03%	41387	4.15%	607279	60.84%	69707	6.98%	0	0.00%	0	0.00%
2005	1050528	275220	26.20%	64898	6.18%	662432	63.06%	47978	4.57%	0	0.00%	0	0.00%
2006	999155	221526	22.17%	61752	6.18%	684062	68.46%	31815	3.18%	0	0.00%	0	0.00%
2007	914504	208229	22.77%	64815	7.09%	619607	67.75%	21853	2.39%	0	0.00%	0	0.00%
2008	730266	148037	20.27%	66985	9.17%	507514	69.50%	7730	1.06%	0	0.00%	0	0.00%
2009	782065	150784	19.28%	80082	10.24%	543717	69.52%	7482	0.96%	0	0.00%	0	0.00%
2010	828697	135284	16.32%	89154	10.76%	592933	71.55%	11326	1.37%	0	0.00%	0	0.00%
2011	1104715	201236	18.22%	118430	10.72%	776199	70.26%	8850	0.80%	0	0.00%	0	0.00%
2012	1468476	236197	16.08%	165517	11.27%	1054852	71.83%	11910	0.81%	0	0.00%	0	0.00%
2013	1798658	274865	15.28%	205596	11.43%	1308937	72.77%	9260	0.51%	0	0.00%	0	0.00%
2014	1784857	268945	15.07%	200874	11.25%	1035314	58.01%	5062	0.28%	274662	15.39%	0	0.00%
2015	1880405	259711	13.81%	205255	10.92%	902175	47.98%	9855	0.52%	503409	26.77%	0	0.00%
2016	1903677	250799	13.17%	208087	10.93%	681812	35.82%	6621	0.35%	728725	38.28%	27633	1.45%
2017	1974711	231972	11.75%	170582	8.64%	577703	29.26%	2620	0.13%	938582	47.53%	53252	2.70%
2018	1948501	232143	11.91%	164059	8.42%	517906	26.58%	2804	0.14%	971810	49.87%	59779	3.07%
2019	464661	86287	18.57%	41595	8.95%	103711	22.32%	322	0.07%	215673	46.42%	17073	3.67%
Total	22817419	4240106	18.58%	2003390	8.78%	12303044	53.92%	480281	2.10%	3632861	15.92%	157737	0.69%

Figure 51 – Submissions to UVa Online Judge

Its use by programming language has been varying over time, the current use being predominantly with C++ and C, and the variation in use can be observed in Chart 3.

[96] https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=23

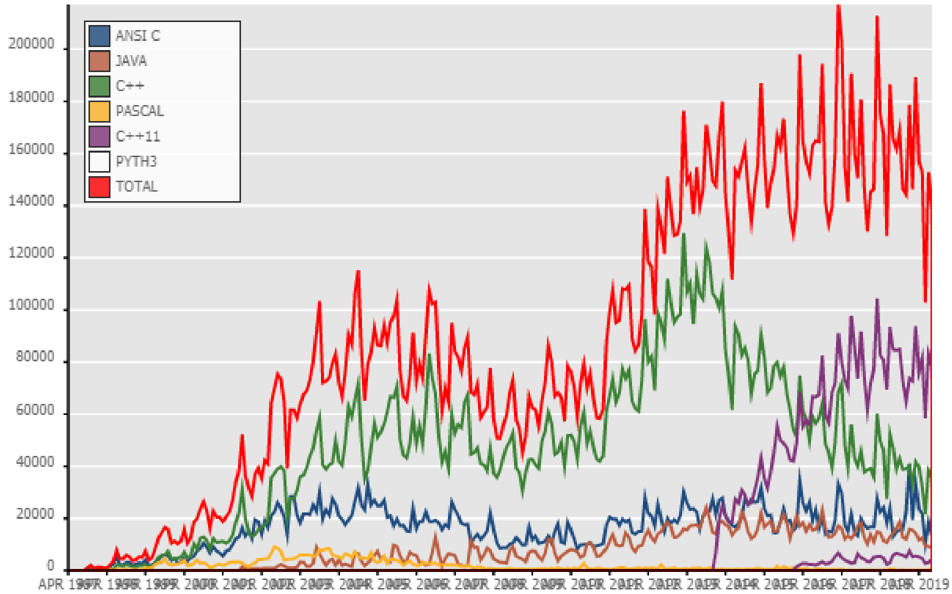


Chart 3 – UVa Online Judge submissions by programming language over time

The system has been accumulating a great amount of challenges, many of them coming from the widely tested ACM contests (Skiena & Revilla, 2003), which can be of great use to teachers.

EduJudge's architecture comprises three types of components (Verdú, et al., 2012):

- learning management system - a learning platform that allows the management of users and resources for pedagogical use, assisting teachers and students and allowing communication between them;
- repository of learning objects - a storage resource that makes it possible to store, share, update, search and manage materials for educational purposes as well as record their metadata;
- evaluation engine - online evaluator with a simple interface that allows program submissions for analysis.

These components are embodied in Moodle (including the QuesTOURnament plugin), crimsonHex and Grape Online Judge, respectively.

QUESTOURnament was developed specifically for integration in Moodle, and consists of an activity and competition management module that aims to encourage students to compete against each other, solving challenges in a set time (Regueras, de Castro, Verdú, Perez, & Verdú, 2009).

CrimsonHex is a public repository within the EduJudge system, possessing a set of properly validated programming challenges, which were stored as learning objects (Queirós & Leal, 2013). The stored programming problems can be searched by different criteria, such as type, author, degree of difficulty or language.

The evaluation engine is supported by Grape Online Judge which automatically evaluates the submitted code for the resolution of a proposed challenge and provides *feedback* to the student. This server results from the evolution of the original UVA Online Judge evaluator, and is able to perform a differentiated evaluation (not just dichotomous right/wrong) and evaluate programming problems of different types, supporting several programming languages, including Pascal, Java, C, C+ (Verdú, et al., 2012).

The process of using the system starts with the teacher creating an activity of QUESTOURnament, adding challenges to the contest by making them available in the repository (Verdú, et al., 2012).

Then students try to solve the challenges, submit the code for analysis and the management system connects to the evaluation engine making the evaluation result available in real time. The teacher can automatically change the rating generated by the system and can also add comments, thus enriching the feedback.

Since EduJudge is widely used in ACM contests, the various possibilities of feedback presented in these contests and generated by the system (Skiena & Revilla, 2003) are presented in Table 13.

Table 13 – Results messages in ACM-ICPC competitions

Result	Description
Accepted (AC)	Congratulations!
Presentation Error (PE)	Your program outputs are correct but are not presented in the specified format. Check for spaces, left/right justification, line feeds, etc.
Accepted (PE)	Your program has a minor presentation error, but the judge is letting you off with a warning. Stop here and declare victory!
Wrong Answer (WA)	Your program returned an incorrect answer to one or more secret test cases.
Compile Error (CE)	The compiler could not figure out how to compile your program. The resulting compiler messages will be returned to you. Warning messages that do not interfere with compilation are ignored by the judge.
Runtime Error (RE)	Your program failed during execution due to a segmentation fault, floating point exception, or similar problem. Its dying message will be sent back to you. Check for invalid pointer references or division by zero.
Time Limit Exceeded (TL)	Your program took too much time on at least one of the test cases, so you likely have a problem with efficiency. Just because you ran out of time on one input does not mean you were correct on all the others, however!
Memory Limit Exceeded (ML)	Your program tried to use more memory than the judge's default settings.
Output Limit Exceeded (OL)	Your program tried to print too much output. This usually means it is trapped in an infinite loop.
Restricted Function (RF)	Your source program tried to use an illegal system function such as fork() or fopen(). Behave yourself.
Submission Error (SE)	You did not correctly specify one or more of the information fields, perhaps giving an incorrect user ID or problem number.

The students can access their grade, and the teacher is responsible for supervising and managing the whole process. Sometimes, to stimulate creativity and participation, it is used as a strategy to suggest to the students the elaboration of challenges to be solved by the colleagues, after review and acceptance by the teacher.

In Figure 52 the structure of the EduJudge system with its components and actors and their interactions are presented.

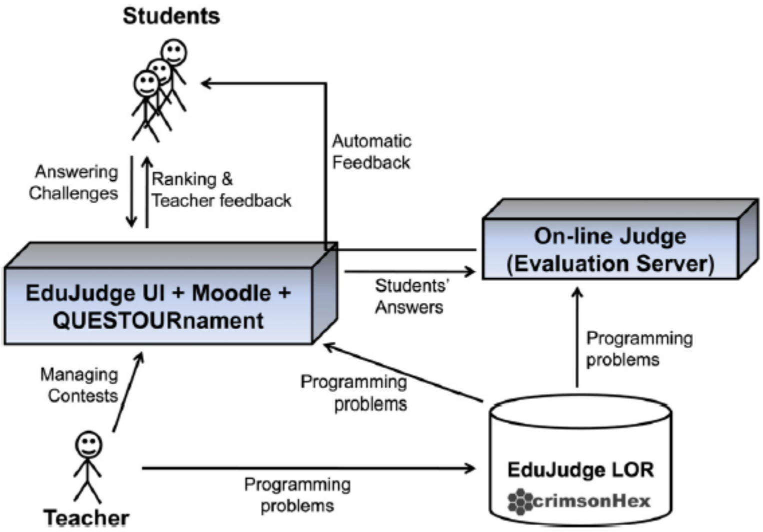


Figure 52 - Structure and interactions of the EduJudge system

(Verdú, et al., 2012)

From the use of EduJugde and the experiences made it was reported that students consider the system useful, promoting competitiveness and creativity, facilitating the learning process. This is evidenced by the results, with better school results in students who used the system compared to others who did not (Verdú, et al., 2012). Also, the teachers point virtues to the system considering that the use of competition tools and automatic evaluation adds effectiveness to the process of teaching and learning programming. However, several possibilities for improvement are mentioned, namely regarding the level of detail of feedback (Wasik, Antczak, Laskowski, & Sternal, 2018).

4.2.3 Mooshak

Mooshak[97] is a system developed by Professor José Paulo Leal, of the Computer Science Department of the Faculty of Sciences of the University of Porto. It is an open source system, in use since 2001, and originally designed to manage online (Leal & Silva, 2003) programming competitions. Nevertheless, boosted by its code evaluation capabilities, it has been increasingly used as a pedagogical tool to support the teaching of programming.

The development of Mooshak (Leal & Silva, 2008) was based on the rules of ACM International Collegiate Programming Contest (ICPC), an annual programming contest, whose first edition took place in 1970[98]. Currently the contest is sponsored by IBM and teams of students from higher education institutions around the world can compete. Meanwhile, with the updates introduced, it is now possible to manage contests from other areas and with different rules, namely those of the Portuguese section of the International Olympiad in Informatics (IOI). It has been used in several programming contests such as South Western Europe Regional ACM Programming Contest (SWERC), Inter-University Programming Marathon (MIUP), Inter-University Programming Tournament (TIUP) or Programming Tournament for High School Students (ToPAS) (Leal & Santos, 2008).

[97] https://mooshak.dcc.fc.up.pt/
[98] https://icpc.baylor.edu/regionals/abouticpc

Among its features are: the availability of problem statements, the possibility of program submission, automatic evaluation of the programs received by the system, querying of submissions made, querying rankings, ranking of users/competitors and global help from the system.

The system allows four different types of users, which are shown in Table 14, as well as their responsibilities and possibilities of acting in the system.

Table 14 – Mooshak users types

User	Actions / responsibilities
Administrator	• Create and manage contests, teams and users • Global coordination of the system
Jury	• Validate, classify and re-evaluate submissions • Provide feedback to competitors • Answer questions from competitors
Competitor	• View problem statements • Submit programs • See results and ratings • Communicate with jury members
Audience	• See contest results

Mooshak also allows the definition of several problems to be solved, and for each problem, in addition to the statement, a set of test cases will be defined. For each of these test cases an input data set and corresponding expected output data will be defined.

Potentially, it is possible to submit problem solving using any compilable programming language (properly installed and configured on the system). To submit the code for evaluation, in the case of Java, the competitor must perform the following procedure:

- create a unique file (.java) with the same name as the main class
- write code
- submit code (not compiled)

It should also be noted that all input data must always be read from standard input and the results presented for standard output, and it is not possible to use packages of graphical environments.

The evaluation will be based on the compilation, without errors, of the source code and, after the execution by the system of the compiled code, by checking the similarity between the output generated by the program and the output defined as expected, based on a set of test data previously configured in the system, when defining the problem (Leal & Silva, 2008).

Mooshak performs the evaluation in two aspects: static and dynamic. The static brokers receive the code and are executed after compilation and may allow unit testing, analysing the code structure or determining engineering metrics from software. Dynamic brokers intervene after each execution evaluating each test case (Leal & Silva, 2008). The goal is to allow standardizing the output in order to simplify its evaluation in cases where the output consists of a set of values.

After the assessment the system generates a response with the result that can range from total success to the indication of errors as well as specification of the type of error. If, for example, no compilation or execution error occurs, but the result obtained is not the same as expected (in at least one of the tests) the system would present the message: Wrong Answer. The possible result messages are shown in Table 15.

Table 15 - Mooshak results messages

Result	Meaning
Accepted	Error-free compilation and all tests successfully verified
Presentation Error	Incorrectly formatted output
Wrong Answer	Wrong result (in at least one of the tests performed)
Output Limit Exceeded	Generated *output* exceeded the space limit
Memory Limit Exceeded	Memory limit exceeded
Time Limit Exceeded	Maximum execution time has been exceeded
Invalid Function	Use of an invalid function
Runtime Error	Program interrupted and an exception is generated
Compile Time Error	Compilation errors
Invalid Submission	Submission error
Program Size Exceeded	Code too large
Requires Reevaluation	The program needs to be reevaluated

A summary table of the results by type of result for each problem can be seen in "statistics" (Figure 53).

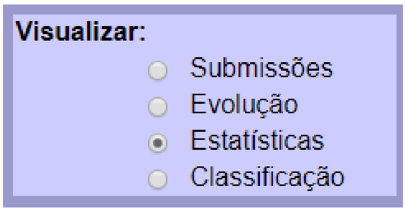


Figure 53 - Choice of information to view in Mooshak

In Figure 54 it is possible to observe an example of submission results.

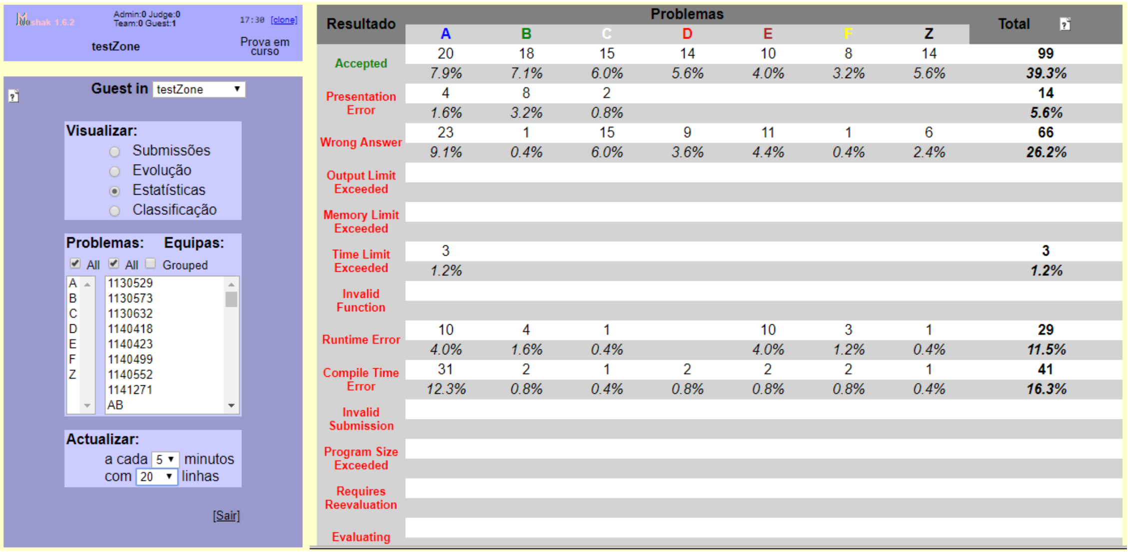


Figure 54 - Example of Mooshak submission results at ISEP

There is also the possibility of alternating between the information to be visualized, being possible to visualize, apart from the statistics, the submissions, evolution and grades, as can be seen in Figure 55.

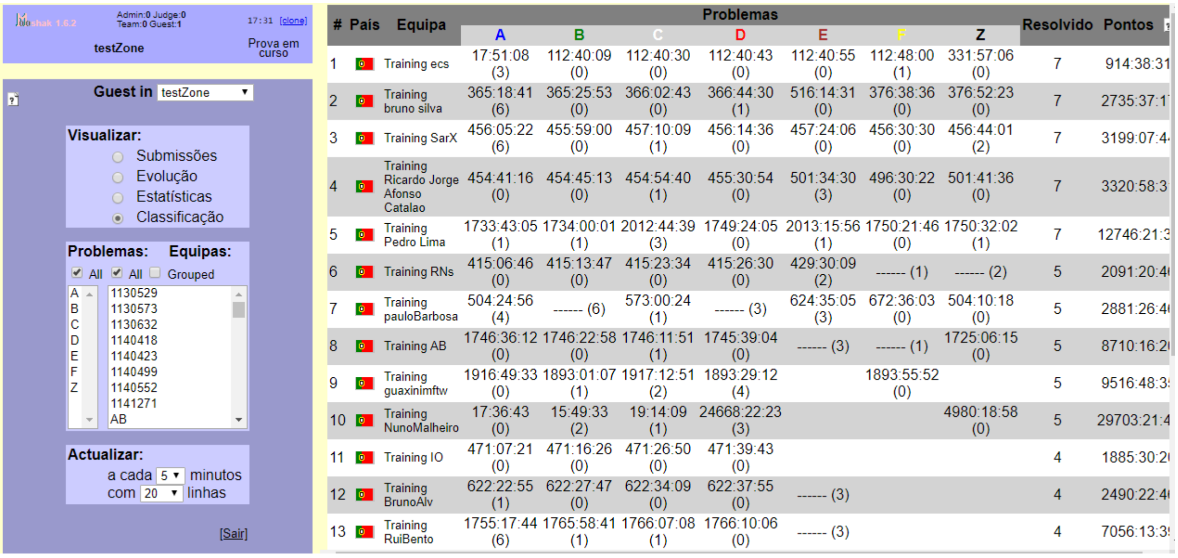


Figure 55 - Example of ratings at Mooshak, ISEP

As already mentioned, Mooshak is a system that has been around for a few years and has been the target of updates and improvements. In its latest version, a diagram editor was incorporated and the possibility of graph evaluation (Correia, 2017) was added.

Besides these tools with a dual purpose (competition and teaching), three other tools were analyzed, developed specifically for teaching-learning activities of the program and presented in the next sections.

4.2.4 BOSS

Bob's Online Submission System (BOSS)[99] (Joy & Luck, 1995) is a system that was designed to receive and analyse programming exercises. The development of the system began in the early 1990s at the Computer Science Department of the University of Warwick, Conventry, in England. In 1993, it consisted of a utility supported by the Sun Solaris operating system having evolved into a graphical environment with connection to a database. New functionalities written in different programming languages were added, which resulted in difficulties in maintenance, portability and evolution of the system.

The evolution of languages, equipment and the increasing demand from users led to the system being abandoned in 1999, and a completely new one being developed. The new version of BOSS was presented in 2000, with new functionalities, namely network operation with increased security, and supporting new programming languages, in particular Java (Heng, Joy, Boyat, & Griffiths, 2005). In 2001, the new version is fully developed and consolidated, incorporating suggestions resulting from feedback from students and teachers who were using the system, being made available as an open source project in order to enhance development by receiving external input. In 2002 it new features are included as follows (Heng, Joy, Boyat, & Griffiths, 2005):

- automation of tests (including unit tests);
- detection of code plagiarism, with the incorporation of the Sherlock component;
- code metrics.

In 2003 a new version is launched, integrating developments resulting from several projects, providing new features:

- new graphic interface;
- access to Boss by a new Java Server Pages application (JSP);
- detection of plagiarism in natural language;
- code plagiarism detection improvements.

In the 2004/2005 school year, the system was tested at the University of Warwick, involving a significant group of students. This test resulted in more than 5500 submissions, with no security breach, but only a few brief and/or minor interruptions.

The BOSS consists of a secure system that allows the submission of student work, performing automatic tests to verify functionality and quality of code along with plagiarism detection. The system can assist the teacher in the evaluation process, as well as being an aid to the student's learning process. This help is provided by the feedback generated by the system, with an indication of the degree of correctness and comments with references to what needs to be amended in order for the program to be improved and to function correctly (Joy & Luck, 1999).

The system was designed to be robust and to work with a large number of students in introductory programming courses, making it easier for students and teachers to interact, thus enabling faster and more effective communication.

In its latest version, the system was fully written in Java and runs on the UNIX operating system, relying on a structure of three servers: students, teachers and tests (Joy, Griffiths, & Boyatt, 2005), according to Figure 56.

[99] https://www.dcs.warwick.ac.uk/boss/

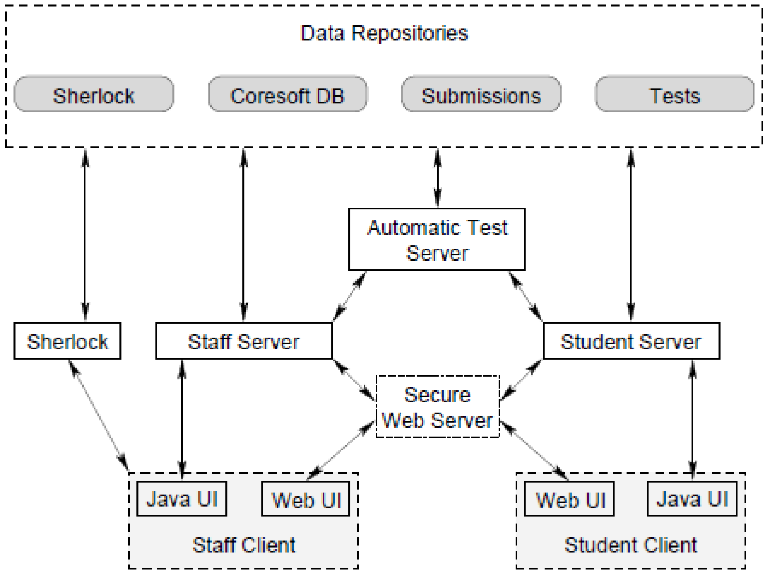


Figure 56 - BOSS System Architecture Overview

(Joy, Griffiths, & Boyatt, 2005)

The student server receives the student registrations and code submissions, which will be tested on the test server by automatically executing the previously defined tests. The teachers' server is used to manage the system and perform the evaluation.

With this distribution of functionalities, advantages were obtained in technical terms, but also in terms of organization, with a clear separation between the roles of the actors in the process, improving security and the overall functioning of the system.

The three-layer architecture of BOSS can be seen in Figure 57.

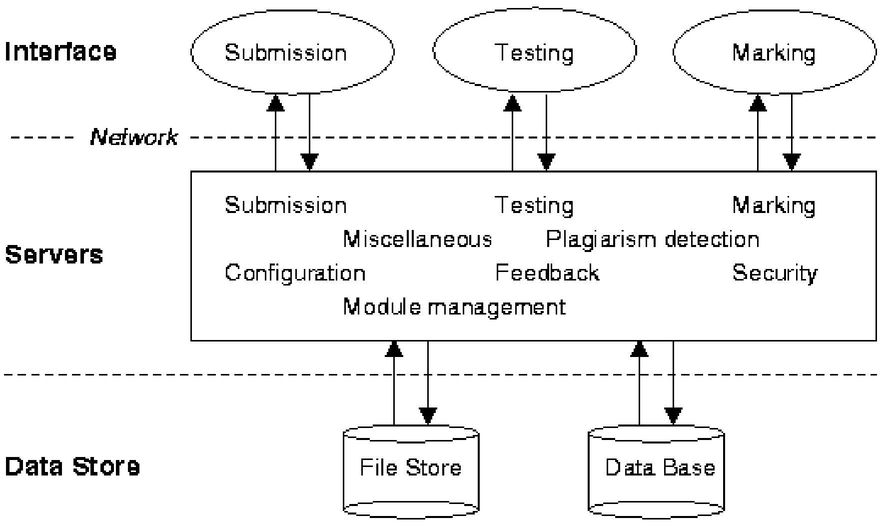


Figure 57 - BOSS - Three-layer Architecture[100]

[100] https://www.dcs.warwick.ac.uk/boss/boss1_architecture.php

From the beginning of its development, it was established that the system should be effective from an educational and pedagogical point of view. Thus, criteria were defined that required the BOSS to have precision, i.e., that its data be reliable. It should also be easy to use and suitable for the purpose of managing and evaluating online code writing work, without negatively impacting the learning process (Joy & Luck, 1995). Finally, it was recommended that it be flexible and capable of evolution, allowing different uses according to the profile, experience, characteristics and interests of the teachers who used it.

The use of BOSS over the years has demonstrated that it is an effective tool that meets the objective of satisfactorily assessing students' submissions, including some pedagogical aspects such as issues associated with plagiarism. The platform-independent client-server architecture gives it a flexible and evolutionary use in both pedagogical and technological terms (Heng, Joy, Boyat, & Griffiths, 2005).

4.2.5 PETCHA

PETCHA is an acronym for Programming Exercises TeaCHing Assistant, a tool whose purpose is to help teach programming (Queirós & Leal, 2012). Among its many features, the one that stands out the most is the automatic correction of exercises, which contributes to its main objective which is to increase the number of exercises to be solved by students (Queirós & Leal, 2012). It allows teachers to create exercises and helps students solve them.

Apart from the evaluation system, PETCHA also has a repository of learning objects, an integrated programming environment and learning management, functioning fundamentally as a coordinating element of e-Learning systems (Queirós R. A., 2012).

The system has been designed to be flexible and to complement existing tools. Thus, instead of having its own development environment it can be integrated with other IDEs having been tested with Visual Studio and Eclipse (Caiza & Álamo, 2013).

For the teacher to make an exercise available to be solved by the students he or she will have to perform three tasks:

- Create the exercise, which consists of defining the expository resources, such as the problem description, and evaluation resources, such as test data or feedback files.
- Store the exercise in a repository.
- Configuring the programming activity in the LMS.

Student action takes place in two phases:

- Select the problem to be solved in the LMS.
- Perform the problem-solving activity using PETCHA together with an IDE.

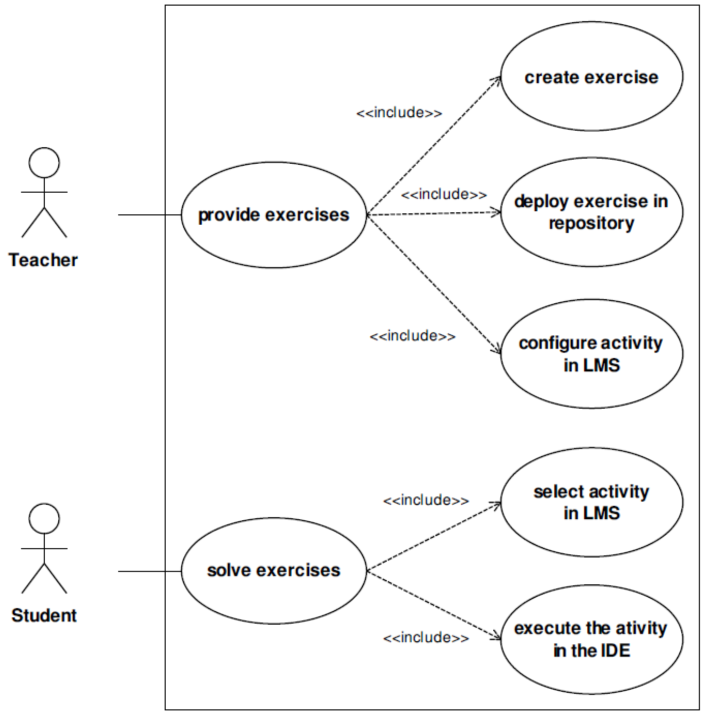


Figure 58 – Actors in PETCHA and their actions

(Queirós R. A., 2012)

The system interacts with teachers and students and there are several shared resources, although the functionalities provided and the graphic interface for each of the roles are obviously different (Queirós R. A., 2012). Both students and teachers need to generate the code and test it in an IDE, as well as verify its operation according to previously defined test cases.

4.2.6 VPL

VPL[101] is a tool developed by the Department of Informatics and Systems of the University of Las Palmas de Gran Canaria, Spain, authored by Rodríguez-del-Pino (Rodríguez-del-Pino, Rubio-Royo, & Hernández-Figueroa, 2011). It is a plugin integrable into Moodle, open source and under the GNU is Not Unix/ General Public Licenses (GNU/GPL license). It was developed in PHP and specifically conceived with the objective of being open source and, in this way, to receive contributions that promote its development.

It is a tool for teaching and learning programming management, supporting a large set of programming languages such as C, C++, C#, Fortran, Haskell, Java, Pascal, Perl, PHP, Prolog, Python and Ruby, and that automatically identifies the language being used.

At its genesis was the intention to provide students with a simple, intuitive and user-friendly environment for code submission and obtaining automatic and immediate feedback.

According to Caiza (Caiza & Álamo, 2013) the architecture of the VPL comprises three main components (Figure 59):

- Moodle module which works as a configuration and administration interface, and also features submission management, evaluation and anti-plagiarism support;
- code editor (Apllet Java) that allows editing, running and testing programs without the need of an installed compiler;

[101] https://vpl.dis.ulpgc.es/

- a Linux server (jail server) to run and evaluate the code, which hosts the environment where the submitted code is evaluated, in a secure way.

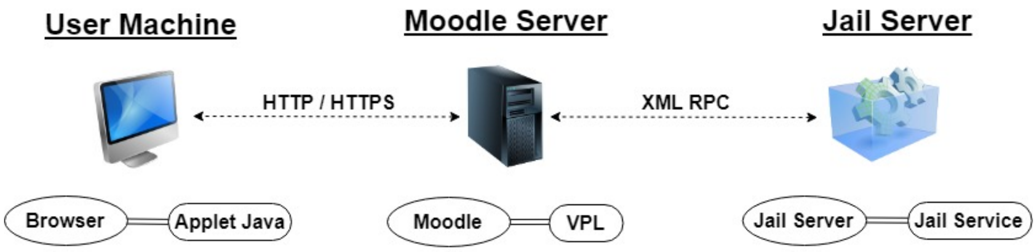


Figure 59 – VPL components and their interconnection

As mentioned, the VPL was written in PHP, however, for the implementation of the jail server, C language was used, being the communication between it and Moodle made by Remote Procedure Call (XML-RPC protocol). The jail system implements a secure remote environment with Chroot Linux and provides services through Extended Internet Services Daemon (Xinetd).

The existence of jail server, in a different virtual machine from the one hosting Moodle, comes from the need to guarantee security to the process, being this a key element of the system. In each execution a virtual user is created, randomly selected. Once the execution is finished, all the files associated with the user in question are deleted.

Thus, from the configuration presented it is possible to see that distributed processing is not possible, since the server responsible for compiling and executing the code is specific and unique. However, given the typical small size of the code to be evaluated and the scarcity of the need for computer resources, especially in beginners, that fact does not seem to be truly limiting.

The VPL has a plagiarism detection feature (Rodríguez-del-Pino, Rubio-Royo, & Hernández-Figueroa, 2012) across source code, looking for similarities between files, having as main objective to detect plagiarism between submissions for the same task by different students of a course. Nevertheless, other sources may also be included, such as submissions for the same task in previous years, or similar code from other sources.

Other relevant features are automatic classification, immediate and automatic feedback and the recording of the history of compilation and implementation of tasks, thus tracking the student's submissions (Thiébaut, 2015). To classify and give feedback, the VPL uses previously defined test cases, specified in a proper syntax and in a specific area. The standard scripts provided by the VPL for program evaluation can be changed in order to improve the evaluation method.

Hence, in the definition of a task to be performed by students, in addition to the problem description and other specifications, it is necessary to specify a set of settings for each activity, among which stand out:

- submission period;
- test cases;
- how the program will be analyzed and evaluated;
- evaluation depending on program size;
- shipping restrictions, such as: number of files, number of submissions, IP address, user profile or resources (execution time, memory, file size and number of processes);
- whether the grade will always be visible to students;

- the possibility of inhibiting the copy/paste resources of the editor, in case it is intended that the programs are typed manually.

As can be inferred from the last point, it is also possible to write the code in an environment other than the VPL editor, for example, in an IDE that the student is already used to, and then upload the code to the system. This is a feature that is not available in several of the similar systems.

Besides allowing the execution and evaluation of complete programs it is also possible to use the VPL for other tasks related to programming. Examples can refer to the visualization and execution of interactive and modifiable examples by the student, or the completion of partially written programs. This type of activity is common in introductory course units and can be very useful to smooth the learning process for beginner students in programming.

It should be noted that, based on the analysis and evaluation in test cases, one of the most relevant and critical aspects in the activities is the formatting of output. In this, as in several similar systems, the result is evaluated by comparing Strings, between the result obtained and the expected result. Thus, in the specifications, examples of inputs and respective outputs should always be given so that students can adapt their code to the specified format.

In Figure 60 we can observe the flow of the automatic evaluation process that occurs in the VPL, from the submission of the work by the student to the obtaining of the result.

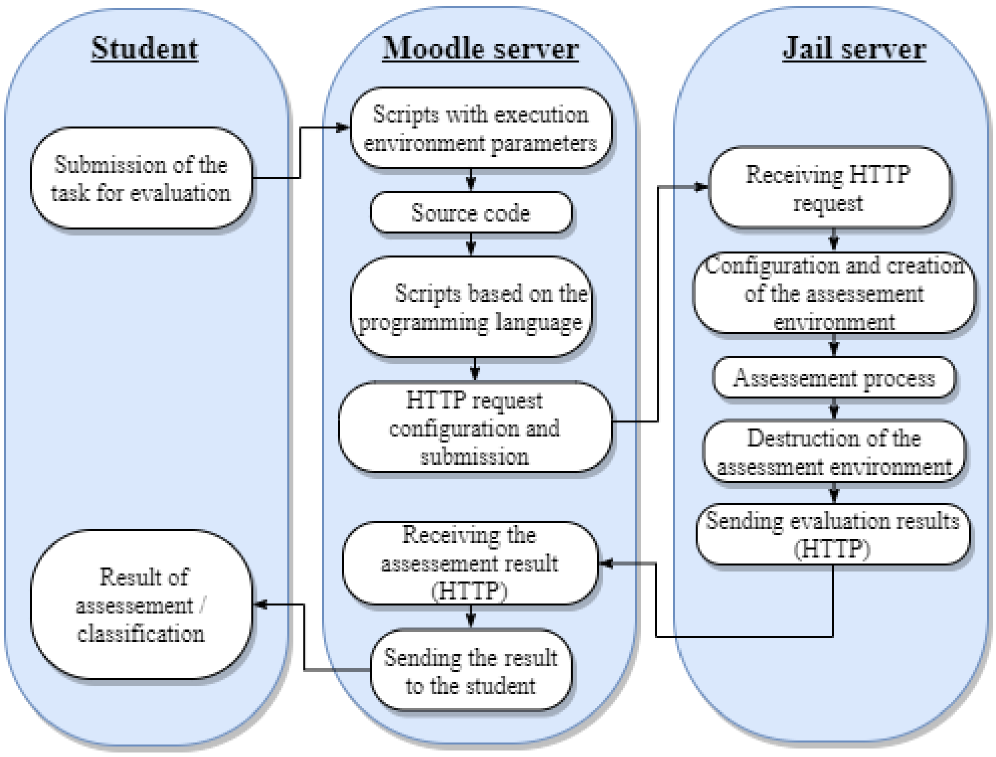


Figure 60 - Automatic evaluation process in the VPL

4.3 THE CHOICE FOR APROG

In the above-mentioned systems, there is a great diversity of features, mode of operation, supported programming languages and purposes. Some allow testing and evaluating complete programs, others only the operation of small parts of the code, others yet, evaluate the encoding style.

From the analysis made and from the observation of the several functionalities made available by the tools, in order to select the one to be chosen for our objective, some criteria were defined by us, being the first one the cost-free. Since this work does not have its own funding, a paid tool would lead to various hindrances, such as obtaining financing and associated bureaucratic issues, in addition to the absence of guarantees for future maintenance.

Another important issue is the possibility of its integration in Moodle, as it is the LMS adopted at ISEP where all the resources of all the courses are hosted and to which the students' access with individual credentials. This solution would avoid the need for becoming familiar with another authentication system as well as the creation of new accounts and new access credentials.

We also looked for a suitable and useful tool to learn how to program. There are many systems that work very well, but they were developed with the objective of managing programming contests, such as the already mentioned Mooshak. It should be noted that Mooshak has been undergoing changes and is increasingly used for teaching programming, in addition to having already been used, on an experimental basis, in APROG in previous years. However, it does not meet the integration requirement in Moodle and new access credentials are required.

It was also intended that the use of a new system should not cause additional difficulties to students, but should be easy to use and allow them to code in their usual environment, so the system should allow the *upload* of the code.

Currently, ethical and moral issues are very important, and plagiarism is a very relevant subject within the academic context. Austin and Brown (Austin & Brown, 1999) consider that the increase in cases of plagiarism is directly linked to the massive availability of information on the Internet, which promotes direct "copy-paste". Technology has facilitated the means of copying, but also the opportunity to develop systems that could counteract it, particularly in the field of information technology. Thus, this concern gave rise to several plagiarism detection systems (Lancaster & Culwin, 2004). In this context another requirement was defined where it was specified that the tool should be provided with similarity detection resources to verify and prevent plagiarism, since this is an important issue for the credibility of institutions (Crittenden, Hanna, & Peterson, 2009).

The tool should also work safely by preventing the execution of malicious software. There are also other safety related issues such as bugs (unintentional) that sometimes cause damage or harm to the functioning of the system by excessive use of its resources (Ala-Mutka, 2005). A typical case is that of programs that execute infinite cycles (they go into loop).

Finally, the tool should support Java, since it is the programming language used in APROG.

In Figure 61 all the requirements defined for tool selection can be observed.

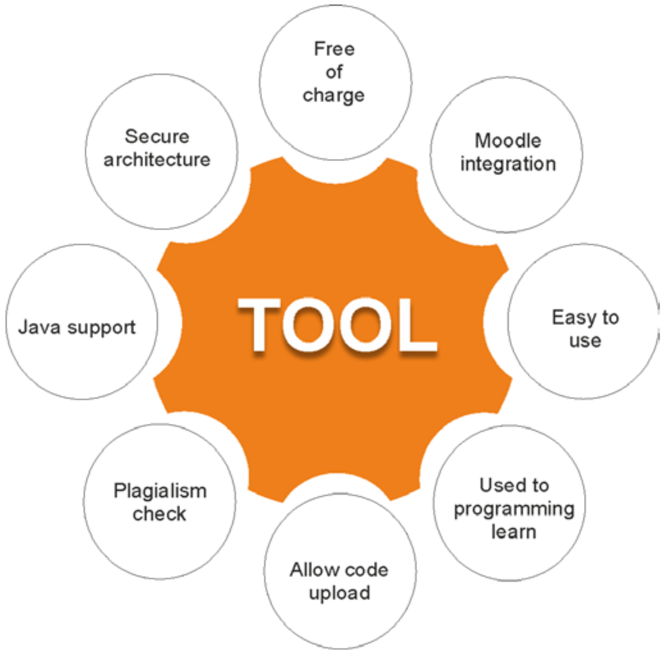


Figure 61 – Tool requirements

From the analysis carried out, the VPL was identified as a versatile tool, free of charge, compatible with existing technical and logistical conditions and potentially effective for the intended purpose.

The main reasons for this choice were its simplicity of use and versatility, the fact that it supports several programming languages (from a future perspective), as well as the existence of more literature and support material on this system than on others. When simplicity of use is indicated, it is meant to refer to its use by the students, since the preparation of the exercises and the configuration of the system, not being complicated, is laborious and time consuming, requiring some effort and patience until it is totally "tuned".

5 VPL IN APROG'S PEDAGOGICAL PROCESS

"A problem is not necessarily solved by the fact that the correct answer has been found. A problem is not truly solved unless the apprentice understands what he has done and knows the appropriation of his actions".

William A. Brownell

This chapter describes how the VPL was used in the context of the APROG pedagogical process.

A contextualization of APROG is carried out within the APROG ISEP-LEI as well as a description of the structure, organization and learning outcomes expected in the APROG CU.

A description of the use of eduScrum is made in the context of APROG, and the operating mode of the LMS adopted at ISEP is also explained, in particular its use in the CU concerned.

The various steps taken to prepare the experience are described, in the bureaucratic, technical, pedagogical and functional dimensions.

The process of implementing the VPL is presented, with a description of the various phases and particularities in each of the school years in which the experiment took place.

Changes and improvements made throughout the process are presented and explained.

At the end of the chapter a review of the process of implementing the VPL is carried out.

5.1 APROG AT ISEP-LEI

The Degree in Computer Engineering (LEI) of ISEP is, in Portugal, one of the most prestigious and sought after, being, typically, among the first three in its area, in terms of the number of annual vacancies[102]. It is, in the context of ISEP, the course with the largest number of students, because, in addition to the two hundred placed annually by the National Competition for Access to Higher Education, it is customary to add a few dozen more from other contingents and, in the various CUs, there are other students who have not obtained approval in previous editions.

5.1.1 Context

LEI is organized in semesters of 16 weeks each, but in most cases in a 12 + 4 organization, the last four weeks being allocated to an integrating CU called Laboratory/Project.

The APROG CU takes place in the first 12 weeks of the first semester of the first year of the LEI and is the first CU where students have contact with programming. One of its main objectives is to promote the introduction to software development.

In the first stage, the purpose is to develop the capacity of logical reasoning through the writing of algorithms and data structuring, which is the process normally adopted in the curricular units of introduction to programming. In the next stage, the knowledge of the logic learned will be applied to a specific programming language. In the case of APROG, the

[102] https://www.dges.gov.pt/coloc/2019/

language adopted is Java, but language is not the central issue, because programming, in addition to the concepts and syntax of the language, mainly requires the ability to think, interpret and design solutions.

In APROG the fundamental concepts related to programming logic, modeling and algorithmic problem solving are taught using the procedural programming paradigm, from an introduction to programming perspective. Under this assumption, no object orientation features of the Java language are used, except for the use of methods embedded in the language.

Although the use of a language suitable for the object-oriented programming paradigm may be questionable in this context, the use of Java aims, at this stage, to allow the easiest and fastest transition from the procedural to the OO paradigm, which is later used in CUs that take place in the following semester to that in which APROG and other curricular units take place throughout the course.

5.1.2 Structuring and organization of APROG

The contents taught in APROG are usually organized in three sequential blocks (Figure 62):

Figure 62 – APROG structural blocks

The first of these corresponds to basic programming concepts, including problem solving, the algorithm and the description of algorithms in pseudocode and using flowcharts.

In the second block we move on to coding in a programming language, starting with the use of simple instructions, and progressively increasing complexity, with the inclusion of decision structures and repetition structures, decomposition into modules, up to the use of indexed structures and files.

The third block is used for the development of an application integrating all the contents taught.

APROG teaching is provided by a team that, in recent years, has had between six to ten teachers for a universe of typically over three hundred students (Table 16).

Table 16 – Number of students and teachers in APROG

School year	2013/2014	2014/2015	2015/2016	2016/2017	2017/2018	2018/2019	2019/2020
Number of students	351	361	383	343	314	307	318
Number of teachers	9	8	8	8	10	6	8

This number of students and teachers implies proper organization and articulation as well as detailed planning in order to ensure equity among the various classes in terms of lessons, achievement of pedagogical objectives and evaluation.

As already mentioned, in APROG the weekly hours of classes are allocated between theoretical (T), theoretical-practical (TP) and practical-laboratory (PL) classes Figure), with the PL-type lessons being divided into two weekly periods of 110 minutes each, as presented previously in Table1.

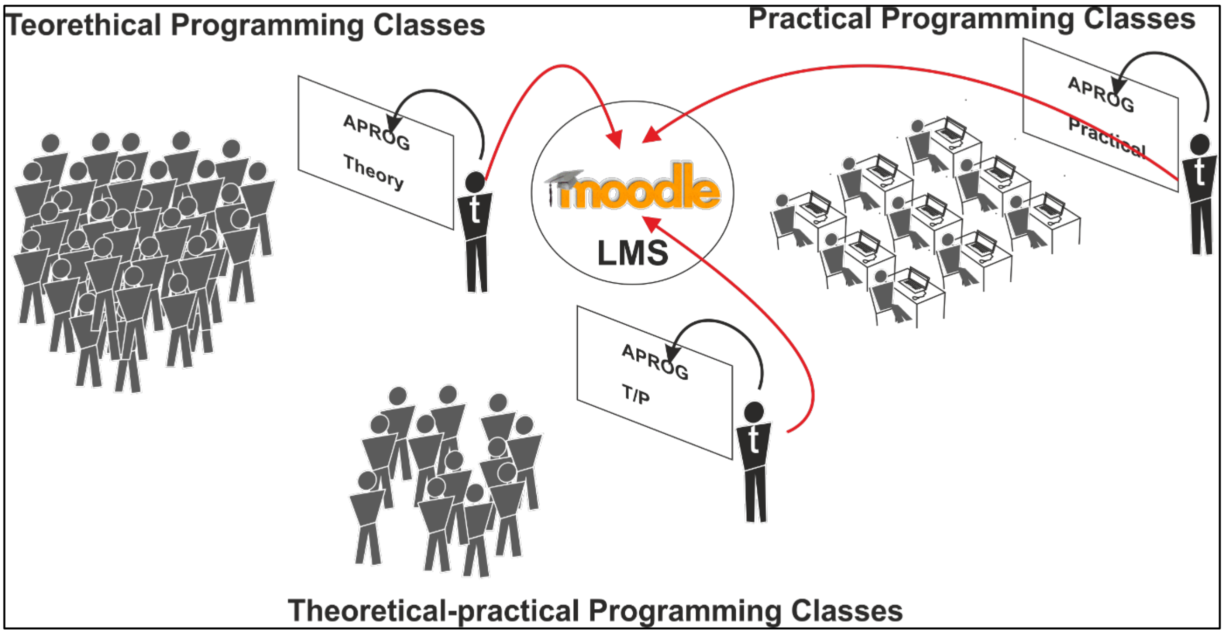


Figure 63 – Types of APROG classes in DEI-ISEP

In APROG classes are organized according to the PL lessons, having, as a rule, up to 20 elements in each class.

In the theoretical lessons the key concepts are presented and the how they can be applied is explained. They are usually taught in auditoriums and comprise four PL classes, which represents an attendance of up to 80 people.

TP lessons usually take place on the same day after the theoretical lesson of the group in question and serve to demonstrate the application of the concepts taught in the theoretical lesson. They are lessons taught for two PL classes at the same time, and each class can have up to 40 students.

The remaining lessons are PL, where the students work, usually in pairs, in classrooms with computers and apply the knowledge of the theoretical and theoretical-practical lessons of that week. They take place in schedules and/or days after the TP lessons, for periods of 110 minutes each and on different days of the week.

5.1.3 Learning Objectives

According to the APROG CU syllabus expressed in the Curriculum Unit Form (FUC) (Annex E), at the end of the process, students should be able to

1. Understand and apply the fundamental concepts of programming;
2. Identify the requirements of a problem, analyze it, create an algorithm for its computational solution and design an appropriate test plan for its validation;
3. Know and understand the Java programming language from the essentially procedural perspective and apply it to the implementation of algorithms, testing solutions using the appropriate test plan;
4. Analyze and design algorithms as models of computational processes structured in modules, create and reuse modules and implement in Java;
5. Know, understand and use indexed data structures as well as manipulate text files;
6. Apply the acquired knowledge to solve real problems and work cooperatively in problem solving and critical analysis.

In order for students to acquire the listed skills and to have the knowledge to do so, APROG's planning establishes the approach of the following topics: programming fundamentals, program coding, data types, modular decomposition, indexed data structures and manipulation of text files.

5.1.4 EduScrum applied to APROG

The purpose of PL classes is training through the resolution of exercises, applying the concepts taught in theoretical classes and demonstrated in theoretical-practical classes.

During these lessons, students should encode the programs in the selected programming language (in this case Java) to solve a set of problems proposed by the teacher and available in Moodle. To code, students use an Integrated Development Environment (IDE) that has, in recent years, been NetBeans.

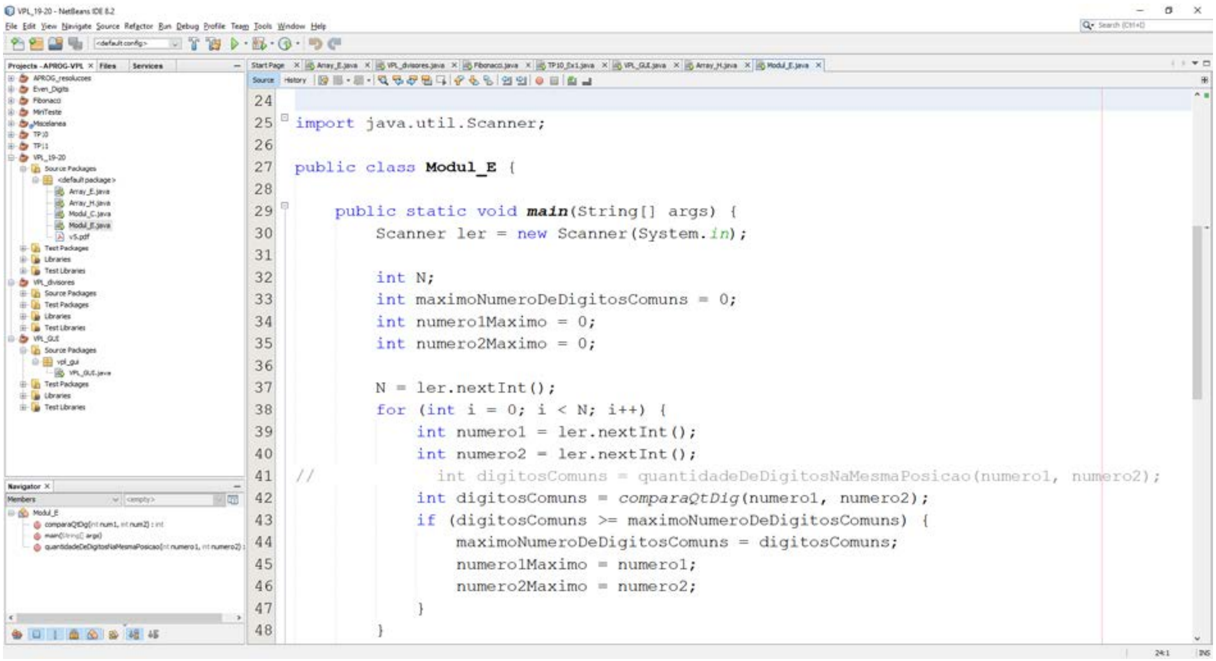


Figure 64 - Example of a program in Java in NetBeans

In recent years, in APROG's PL lessons, the eduScrum (Delhij, van Solingen, & Wijnands, 2015) methodology has been used, with students organized in teams of two (exceptionally three), where they follow a set of procedures focused on the needs and instructions of a Product Owner, as presented in section 2.4.3, here depicted by the teacher.

The use of eduScrum aims to bring the academic and business environment closer together, since Scrum is a methodology widely used in the development of software (Moniruzzaman & Hossain, 2013). It also aims to promote collaborative work and the development of soft skills, fostering autonomy, responsibility and self-confidence in students.

Each eduScrum team has a board (Figure 65) where the tasks to be performed, to whom they were assigned and the status of each task are recorded, allowing, at each moment, a clear view of what has already been done and what remains to be done. This board is an essential and mandatory tool and should always be updated and available to all team members and the teacher, with each change being recorded immediately after it is made, in order to provide an overview of the project status.

BOARD TURMA: GRUPO:				
Tarefas	Atribuida a	Em curso	Realizada	Aceite pelo Prof
TASKS	**Assigned to**	**In progress**	**Done**	**Accepted**
BLOCO 1 - Semana 26 a 30 de Set				
Exercício1 -A				
Exercício1-PeerReview-B				
Exercício2 - B				
Exercício2-PeerReview-A				
Exercício3 - A				
Exercício3-PeerReview - B				

Figure 65 - EduScrum board example

The organization of the project is based on the peer review model, with each task being reviewed by a colleague before being analysed by the teacher.

Work is structured in sprints, generally of equal duration, throughout the project, in recent years three sprints being considered, each having a duration of four weeks.

It is necessary to plan the sprint by holding a meeting to define and clarify the goals of the sprint and all subsequent sessions should begin with a short stand-up meeting (daily stand-up meeting) as mentioned in section 2.4.2.4.

In the first meeting of the week the work to be done should be planned, stating:

- What is expected of the group: learning objectives and evaluation acceptance criteria;
- Exercises to be solved in the week (each task with an indication of the effort units), in what order and by whom, balancing the effort units per student.

Students should begin their work according to *board* planning and records.

At the beginning of the second lesson of the week, at the daily stand-up meeting, each student must answer:

- What was done?
- What am I going to do today?
- Is there any blockage?

After completing a task and its verification by a colleague, the teacher can question the members of the group in order to assess whether the work was carried out correctly and whether the students understood the solution. If the task has errors or students cannot fully explain it, the task will not be accepted, and students will have to redo it.

At the end of each sprint, there is a review of the sprint, where students will individually solve a set of exercises. Up to one week after the review of sprint, the teacher should disclose to students the results of the sprint assessment, with the final grade being obtained as follows:

- 30% assessment of sprint tasks (weighted average of tasks successfully performed);
- 70% of the individual evaluation.

After that, students usually have a moment to look back on sprint and make a personal reflection.

5.1.5 The basic support LMS for APROG operation

At ISEP, Moodle has been used for more than a decade as an LMS to support teaching activity and the pedagogical needs of students.

When each new student joins ISEP, they will be given access credentials to the Portal and to Moodle, which will serve for the entire period in which they remain as students of the institution. Access to the various areas and materials is managed by the person in charge of each course unit attended, and other areas may also be granted upon request. The regular use of this LMS in all courses has allowed to standardize procedures and ways of providing content.

The relatively long experience of using Moodle at ISEP has boosted its growing use by teachers and students, exploring much of its basic potential. The familiarity with the process and the knowledge acquired led to the appearance of new needs and the design of new functionalities, which resulted in the installation of additional plugins, namely:

- Switch to full screen - atto_fullscreen
- Progress Bar - block_progress
- BigBlueButtonBN - mod_bigbluebuttonbn;
- Bulk meta course link - enrol_metabulk
- Course file area - repository_coursefilearea
- jQuery - local_jquery
- Bulk Meta-disciplines - local_bulkmeta
- Questionnaire - mod_questionnaire;
- Quickmail - block_quickmail
- RecordingsBN - mod_recordingsbn;
- Reorder disciplines - local_resort_courses
- Virtual programming lab - mod_vpl

The following new modules were also developed by the technicians responsible for Moodle at ISEP:

- ISEP Subjects - block_course_list_isep
- Subject Information - block_info_disciplinas
- Portal Integration - block_integracaoportal
- Pedagogical Questionnaires - block_limesurvey
- Document Printing - block_reprografia

The internally developed modules aimed at filling some gaps and adapting the operation of Moodle to the specific needs and mode of operation at ISEP.

Students enrolled in the APROG CU are automatically granted access to their Moodle area. The use of Moodle in APROG main purpose is to provide all the information related to the CU, to serve as a means of communication between teachers and students, sending messages or publishing announcements, as well as allowing the submission of works for assessment.

In Figure 66 the homepage of APROG for the school year 2018/2019 is shown.

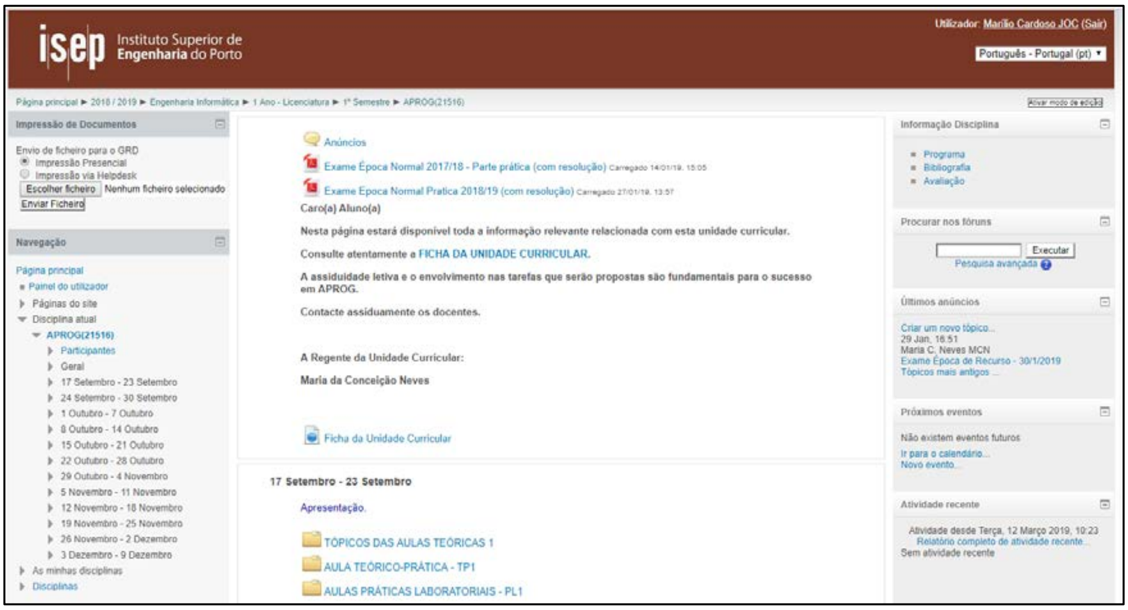


Figure 66 – APROG on Moodle

The contents are organized by weeks and, in each week, the topics to be dealt with are indicated and the contents are available for each type of lesson (Figure 67).

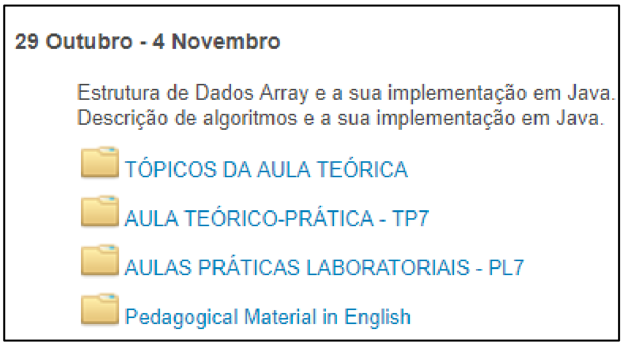


Figure 67 – APROG materials in Moodle for a specific week

For PL lessons, a .PDF file containing a set of exercises that the students must solve and whose resolutions must be analyzed by the teacher is available each week (until the eighth).

The exercises proposed for resolution were previously solved by the teachers in order to verify their feasibility, size, level of difficulty and use of resources and concepts that are intended to be practiced.

As an example, in Figure 68 the student's view for the download of PL7 exercises is shown.

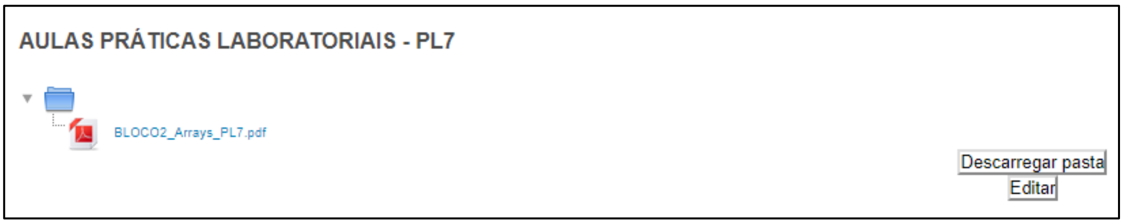


Figure 68 - Exercise sheet for download of PL7 class

Due to the fact that there are some foreign students (in ERASMUS mobility) some translations in English of the different materials are also available in a specific folder (Figure 69).

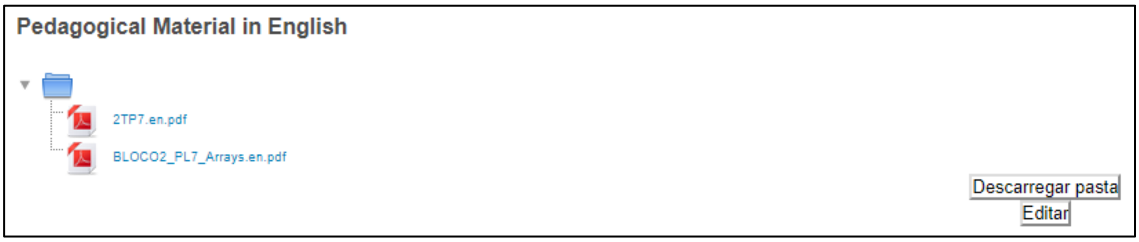


Figure 69 - Material in English for download

The number of exercises in each form varies according to the size and complexity of each one and depending on the subjects involved.

Since there are students with very different rhythms and backgrounds (previous knowledge), the exercises are divided into two sections, the second entitled "Complementary Exercises".

This strategy has been adopted in recent years, being adjusted and appropriate to the motivation and stimulation of students regardless of their knowledge and pace of learning, allowing those who advance more quickly to continue training and evolve without others feeling discouraged or frustrated by not being able to solve the same amount of exercises as their colleagues.

Table 17 shows the number of exercises of each type in each of the weeks of coding in Java based on worksheets, corresponding to the second block, as mentioned in section 5.1.2.

Table 17 - Number of exercises per worksheet and type

	Worksheet	PL5	PL6	PL7	PL8
Number of exercises	base	10	9	6	5
	complementary	4	9	6	2
	total	14	18	12	7

In each exercise, the level of difficulty/complexity is marked and a notation with the symbol (*) has been adopted. The scale goes from (*) - simpler, to (***) - more complex, as can be seen, for example, in Figure 70 concerning exercises 6 and 7 of PL6.

Exercício 6 (*)

Faça um programa que permita determinar volumes de sólidos de revolução (cilindros, cones e esferas). Para cada sólido será introduzido o tipo de sólido e as respetivas dimensões. O programa termina quando o tipo de sólido for a palavra "FIM". Implemente o programa de forma modular.

OBS: $V_{esfera} = 4/3\ \pi\ R^3$
$V_{cilindro} = \text{Área Base x Altura} = \pi\ R^2\ \text{Altura}$
$V_{cone} = 1/3\ \pi\ R^2\ \text{Altura}$

Exercício 7 (***)

a) Faça um módulo que verifique se um número é ou não um número octal.
b) Faça um módulo que converta um número octal em número decimal.
c) Faça um programa que leia uma sequência de números na base octal e os converta em números decimais. A sequência termina quando for introduzido um número que não é octal.

Figure 70 - Exercises 6 and 7 from PL6 - 2018/2019

5.2 CONDITIONS OF IMPLEMENTATION AND INSTALLATION OF THE VPL

To use the VPL, and after some initial meetings, the needs related to: material, technical and human resources were identified, so it was necessary to centralize efforts to secure these before the implementation that would support this study.

It was also necessary to verify compliance with the legal requirements for the operation of LEI in general and of APROG in particular, namely with regard to pedagogical issues and, more specifically, regarding the evaluation process.

5.2.1 Preliminary Steps

For the installation and use of the VPL it was necessary to verify whether it was feasible in the context of ISEP and to identify and guarantee conditions so that its use could take place without constraints.

For this purpose some preliminary experiments were carried out between June and July 2017 that included the installation of Moodle and the VPL on a specific virtual machine, as well as an installation of another server necessary for the operation of the system on a virtual machine other than Moodle.

Three users with three different profiles were also created: administrator, teacher and student.

After this installation, a VPL activity was created, related to a simple exercise, and its functionality in Python and Java was verified.

5.2.2 Bureaucratic issues

Since it was intended to use ISEP's institutional Moodle, it would be necessary to obtain permission to install the VPL plugin. Thus, on 28th July 2017, a petition was addressed to the presidency of ISEP, which was assigned the number 33551, having received a favorable ruling on the 2nd August 2017 (Annex A).

Also with regard to pedagogical, operational and legal issues, the director of the course of LEI and the head of the curricular unit (RUC) of APROG were approached on the 1st August 2017 in order to obtain authorization for the experiment, addressed by e-mails with the request for such authorizations, so as to document the procedure. Both requests were answered positively, as can be seen in the e-mails presented in the Annexes (Annex B and Annex C).

In the school year 2018/2019 a new course direction was appointed and the request for authorization to repeat the process was reiterated. The request was made verbally and was accepted without restrictions by the new director of the LEI.

In the school year 2019/2020 a new APROG RUC was appointed and was contacted in order to maintain the use of the VPL in the model previously used, responding positively to this request.

5.2.3 Technical issues

Once the formal authorizations were obtained, the plugin was installed in ISEP's institutional Moodle. The installation was executed by the ISEP technician responsible for the administration of Moodle with monitoring and supervision by the author of the study, and was completed on 6th October 2017 (Figure 71).

Figure 71 – VPL installation in ISEP Moodle

Nevertheless, some deficiencies were detected, namely the accesses due to the functioning of the firewall, which were only definitively solved on 20th October 2017.

The VPL greatly values security and for this purpose it is advisable to install the VPL plugin and jail server separately so that the compilation can be done in isolation and on a different machine from the one hosting Moodle[103]. Thus, in each execution, a virtual user is selected at random and all the files associated with that user are destroyed at the end of the process.

For a proper and efficient use of the VPL it is necessary to take into account the use of resources according to the expected needs, particularly with regard to the execution time of a program so as not to degrade the performance of the server. Limits must also be set for other parameters such as the memory to be used, the number of processes or file size (Figure 72).

[103] https://vpl.dis.ulpgc.es/index.php/support

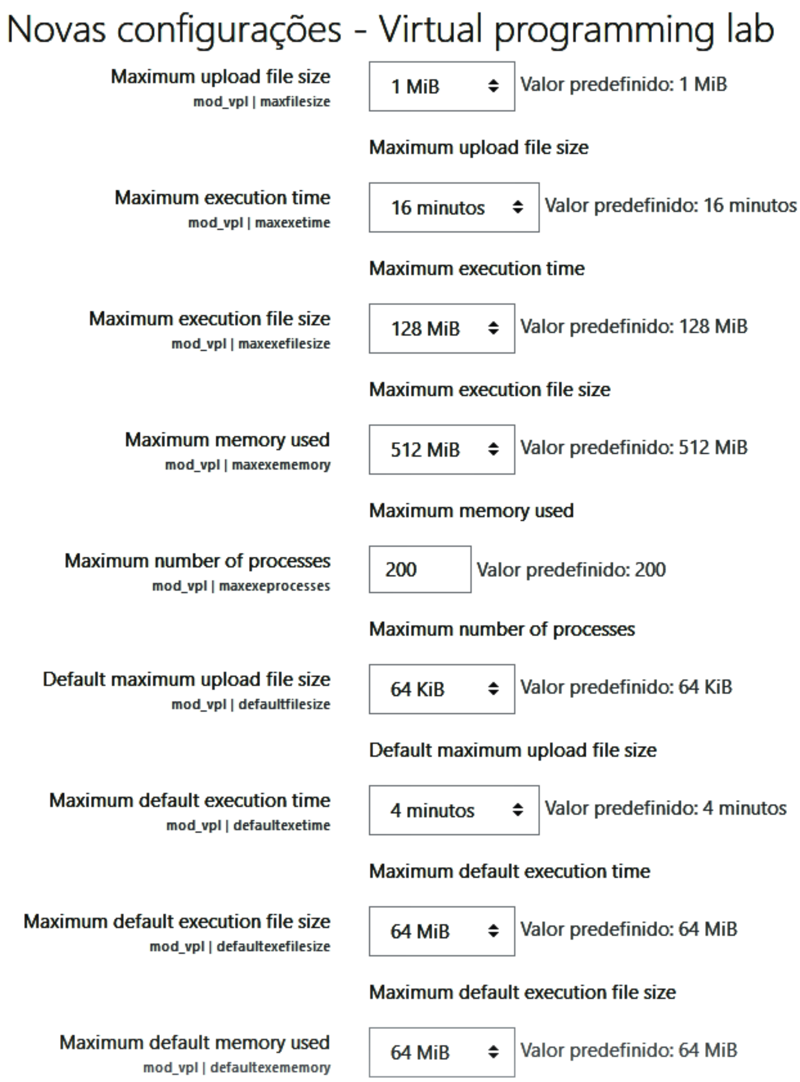
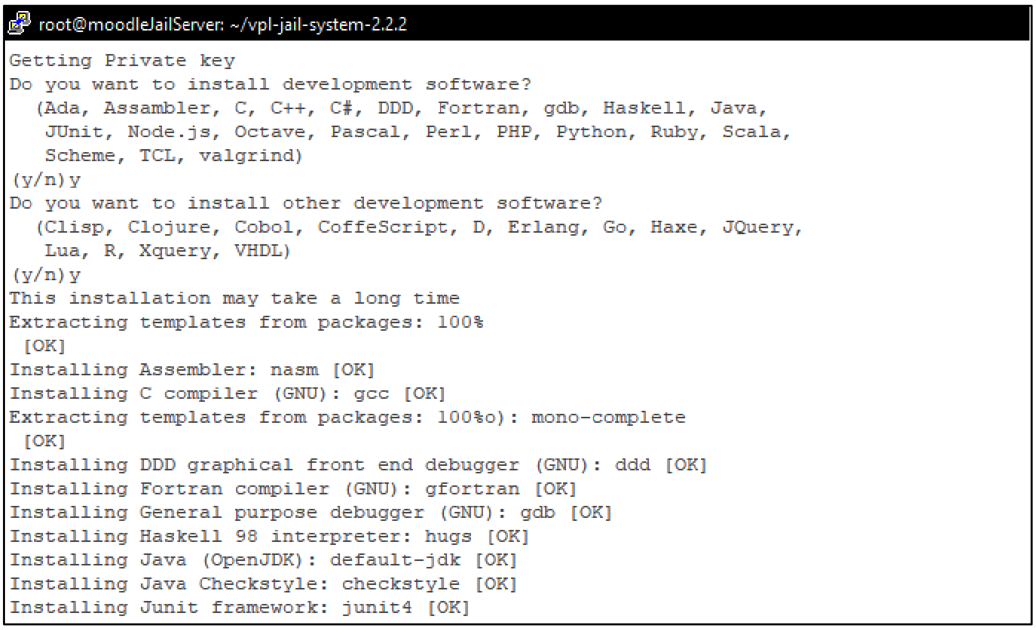


Figure 72 – Configuration of parameters in the VPL

It was also necessary to install compilers for the programming languages that were intended to be used with the VPL. In Figure 73 the screen of the installation of some compilers done at the VPL at ISEP is shown.



```
root@moodleJailServer: ~/vpl-jail-system-2.2.2
Getting Private key
Do you want to install development software?
   (Ada, Assambler, C, C++, C#, DDD, Fortran, gdb, Haskell, Java,
    JUnit, Node.js, Octave, Pascal, Perl, PHP, Python, Ruby, Scala,
    Scheme, TCL, valgrind)
(y/n)y
Do you want to install other development software?
   (Clisp, Clojure, Cobol, CoffeScript, D, Erlang, Go, Haxe, JQuery,
    Lua, R, Xquery, VHDL)
(y/n)y
This installation may take a long time
Extracting templates from packages: 100%
 [OK]
Installing Assembler: nasm [OK]
Installing C compiler (GNU): gcc [OK]
Extracting templates from packages: 100%o): mono-complete
 [OK]
Installing DDD graphical front end debugger (GNU): ddd [OK]
Installing Fortran compiler (GNU): gfortran [OK]
Installing General purpose debugger (GNU): gdb [OK]
Installing Haskell 98 interpreter: hugs [OK]
Installing Java (OpenJDK): default-jdk [OK]
Installing Java Checkstyle: checkstyle [OK]
Installing Junit framework: junit4 [OK]
```

Figure 73 - Installation of compilers at the VPL.

In parallel, it was necessary to check the remote access conditions, access permissions and network settings as well as firewalls.

In the case of the installation made, some initial difficulties of external access were identified. At this point it was found that access was only possible through the VPN (Virtual Private Network) of the DEI which was highly limiting for widespread use from the outside. Figure 74 shows the error that occurred during the access attempt.

Figure 74 - Jail server external access error

It was therefore necessary to review the settings of firewalls to allow access.

The installation was carried out on a Linux server with no graphical component. Although the VPL allows the use of components of graphical environments, for this to be possible, the server must support this functionality, and for this purpose the X11[104] protocol must have been installed, which in our case was not possible due to the characteristics of the server.

A test was performed with a java program, with Swing[105] components, and the error shown in Figure 75 was detected.

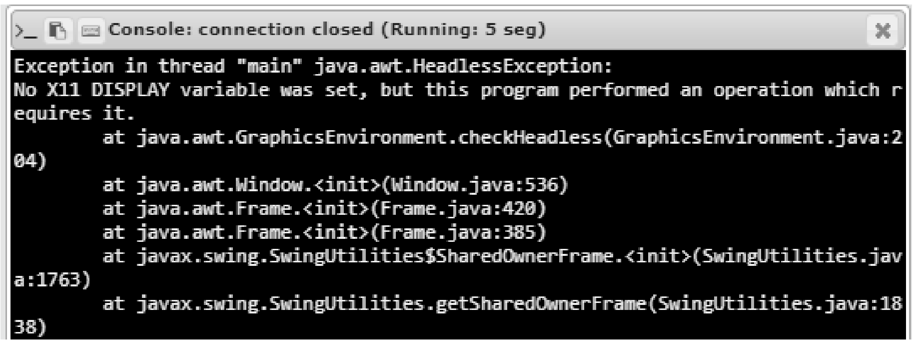


Figure 75 – Error of absence of graphical environment

Besides X11, it would be necessary to use the VNC protocol (Virtual Network Computing) to allow the visualization of graphical interfaces remotely, which would add complexity and, eventually, less security to the system. Thus, the possibility of using graphic components was inhibited.

Also with regard to access, the first time the connection to the jail server is made, it would be necessary to accept the security certificate (Figure 76).

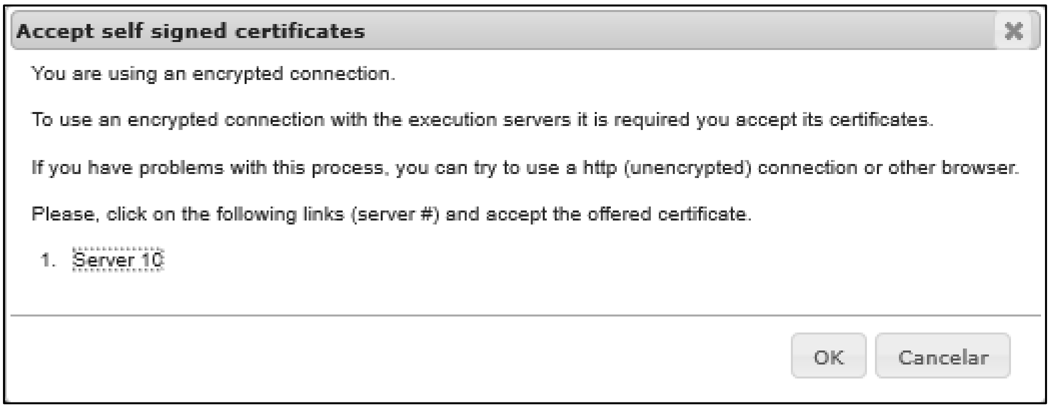


Figure 76 – Jail server security certificate

After this stage, some functional tests were performed, moving on to the next step related to the clarification of the pedagogical aspects that are presented and discussed in the next section.

5.2.4 Pedagogical issues

One of the main objectives of the process was to assess the possibility of contributing to making learning more autonomous, therefore the use of the system should not be too complex.

---

[104] X-11 is the standard toolkit and protocol for GUI on Unix and similar systems, such as Linux

[105] Java High Level Graphics API

Autonomy is not, however, synonymous with unsupervised work, with the teacher continuing to play a key role in monitoring and supervising students' work, identifying shortcomings as early as possible and enhancing the acquisition of knowledge in a sustained manner. In fact, the unsupervised use of tools of this type can be counterproductive, since, once the correct result is obtained, the student can presume that the resolution has been developed in the best way, which may not be true.

The experience associated with this study was welcomed by APROG's RUC, although the following conditions were imposed:

- The existing process was not to be disturbed;
- Ensuring that it would not have a negative impact on the current teaching process;
- Complying with the FUC planning of the CU in the pilot classes;
- Ensuring that there are no differences in assessment between students who used VPL and those who did not;
- Participation of students on a voluntary basis.

These conditions were accepted and fully complied with by the author of the study and those involved in the experiment.

So as to make the process more fluid and transparent, the intention to carry out the experiment was communicated to the colleagues of the APROG teaching team, in order to increase critical mass and extend the experiment to a significant number of classes/students.

A plan for the implementation of the VPL was outlined for use in the context of APROG that considered the planning of the CU, the pace of content progress and the suitability of the exercises to be used. This plan aimed at not disturbing the normal functioning of the classes (as required by the RUC) and, as a new way to learn / practice in a context of new concepts and pedagogical tools, it would be necessary to promote and ensure a smooth adjustment so that students could easily adapt to this new concept.

In this initial stage of learning, the impossibility of using graphic components was not, from the pedagogical point of view, a problem.

The focus of the learning being programming logic and language syntax, the introduction of a graphical environment would not be paramount and would only increase complexity.

5.2.5 Functional issues

As mentioned previously and shown in Table 11, the automatic code evaluation tools can support one or more programming languages. One of the initial assumptions at this level was that the tool to be adopted should allow the use of the Java language since it is the programming language used in APROG.

The VPL allows the use of a large set of programming languages, according to the compilers installed. The compiler selection is made automatically and according to the source code file extension.

In the case of APROG, Java being the programming language used, it was intended to limit the use of the VPL to that language only. Thus, in the running definitions (file *vpl_run.sh*) the Java compiler was explicitly invoked, as can be seen in the example of Figure 77.

Execution files

vpl_run.sh

```
#! /bin/bash
cat > vpl_execution << 'EOF'
#! /bin/bash
javac -J-Xmx128m AprogAv2.java
java AprogAv2
EOF
chmod +x vpl_execution
```

Figure 77 - Example of *vpl_run.sh* with invocation of Java compiler

This implied that the class name (and the .java file) coincided with the name defined in the execution files. If this requirement was not met, it was not possible to run the program, and the error shown in Figure 78 is obtained.

```
Console: connection closed (Running)
javac: file not found: AprogAv2.java
Usage: javac <options> <source files>
use -help for a list of possible options
Error: Could not find or load main class AprogAv2
```

Figure 78 - Compilation error in running attempt

If the evaluation option is executed, the error obtained is the same, appearing in the area related to the test result (Figure 79).

```
--- Program output ---
javac: file not found: AprogAv2.java
Usage: javac <options> <source files>
use -help for a list of possible options
Error: Could not find or load main class AprogAv2
```

Figure 79 - Compilation error in evaluation attempt

There are also other restrictions to be considered, namely the non-use of packages which are often automatically created in several IDEs. Thus, to be able to use the code from an IDE, it was necessary to remove or comment on the line related to package, should it exist.

Another restriction had to do with the use of components from graphic environments. Although this use is possible it can complicate the process.

Other aspects have also been observed, such as, for example, the definition of the decimal separator as ".".

A crucial factor for the success of the process is the formatting of output, which will have to match exactly what is specified because, as already mentioned, the success check is based on the comparison of Strings.

After these initial steps (checking the functionality and testing the concept, creating the technical, logistical, pedagogical and legal conditions), it was decided to move on to the implementation of the experiment using the VPL within the APROG practical-laboratory classes, as presented in the following section.

5.3 VPL PREPARATION IN THE CONTEXT OF APROG

At the beginning of this stage, a reflection was made on which exercises, from the list of existing exercises in the APROG sheets, should be used in the VPL. Since it was not allowed to interfere with the existing teaching-learning process in APROG, the number of exercises should be narrowed down and those exercises with the greatest potential for use should be identified. This analysis was carried out based on the importance of each exercise and how easily it can be adapted to the VPL with the minimum of changes and without distorting the context and its objective.

Six programming exercises were selected for the case study, two of each week, involving the elementary use of the main programming elements. The exercises chosen were PL6 numbers 3 and 5, PL7 numbers 2 and 4 and PL8 numbers 2 and 3 (Annex A and Annex J).

The choice of exercises took into account the various contents taught, namely, the presence of declarations, decision structures, repetition structures and embedded structures, modular decomposition and the use and manipulation of indexed data structures (one-dimensional and two-dimensional arrays, i.e., vectors and matrices).

Considering that the methodology used in the APROG CU is eduScrum and based on sprints, the moment when the use of the VPL should start was also analysed, and it was decided that it should only occur in the second week dedicated to coding in Java.

This decision was due to the fact that it was understood that, if there were new contents and tools, unknown to most students, such as the Java language itself and the IDE to be used, introducing another tool at the same time could cause some confusion and demotivation which could become counterproductive. So, after the first contact with Java and the proposed IDE (NetBeans), students would be more comfortable initiating contact with a new context, with the advantage of coding in the IDE at an early stage and only then loading the code into the VPL for validation.

In Figure 80 we present the various stages of preparation of the process, from the initial tests, the bureaucratic, technical, pedagogical and operational issues to the definition and implementation of activities in the VPL.

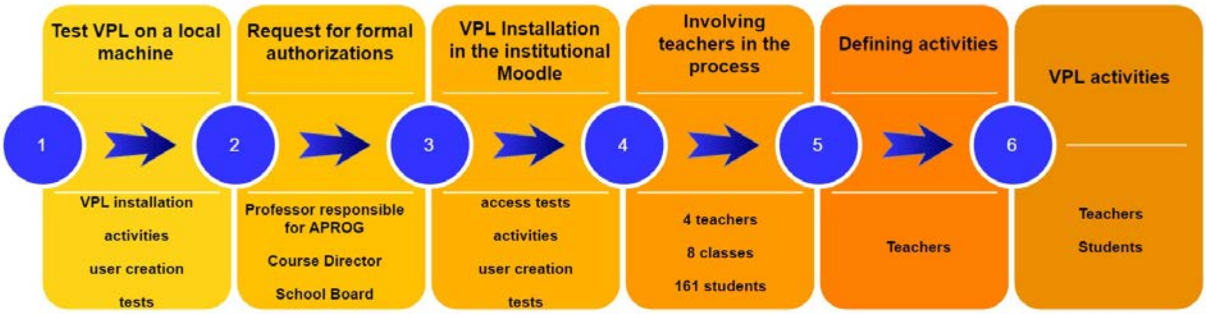


Figure 80 – Stages in the planning process

5.4 VPL IMPLEMENTATION IN THE CONTEXT OF APROG

For this experiment, some pilot classes were chosen, the number of which varied throughout the various editions of the APROG CU, according to the teachers who taught the PL classes and who were willing to collaborate in the experiment.

5.4.1 Introduction

For the VPL activities implementation a CU was created in Moodle (APROG-VPL) in order to meet all the requirements and respect the agreed conditions, not interfering with the content organization and normal functioning of APROG, thus enabling some experiments without the risk of any possible error impacting APROG.

Since not all APROG students would use the VPL, only those involved in the experiment would have access to APROG-VPL and would specifically be enrolled for this purpose. By already having credentials and experience in accessing Moodle, the process has become quite simplified for the students.

As mentioned, one of the conditions imposed was the student's voluntary membership, so even if the student belonged to a class that would use VPL and was therefore enrolled in APROG-VPL, the student might not be able to submit the work developed.

5.4.2 Creating activities in the VPL

In this section we will address the creation of activities in the VPL from the perspective of the teacher.

VPL being a Moodle plugin, the creation of an exercise in the VPL by the teacher editor starts by adding an activity to Moodle (Figure 81).

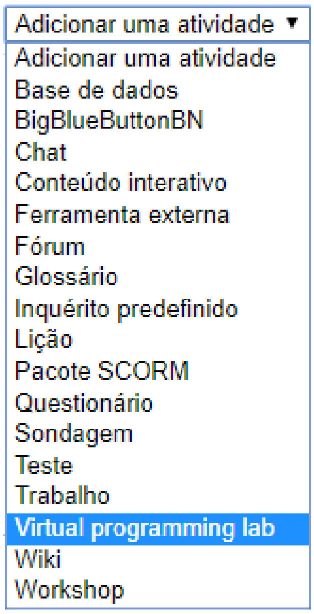


Figure 81 – Adding a VPL activity to Moodle

After choosing this option (Virtual Programming Lab) the configuration and parameterization options are available (Figure 82) where the activity to be performed will be completely defined.

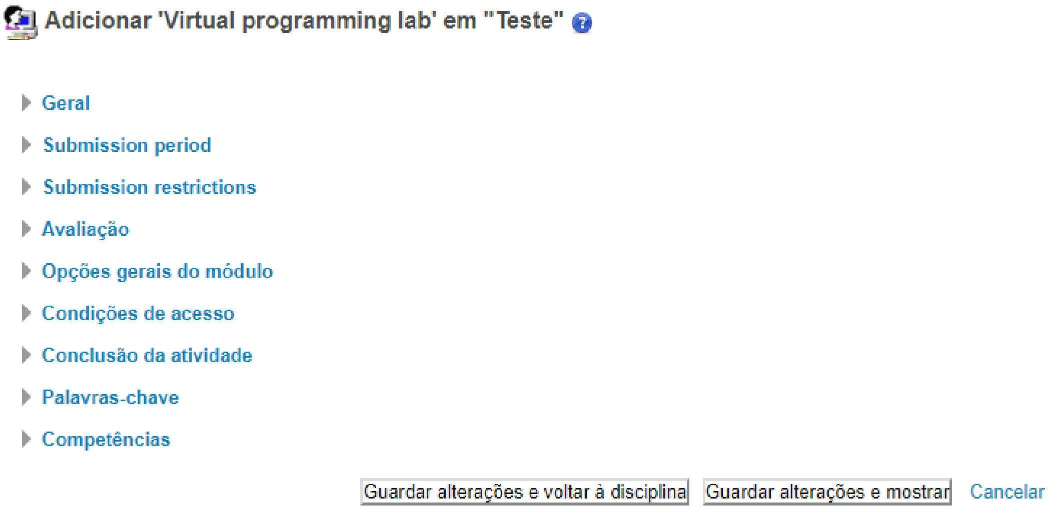


Figure 82 - Parameter setting of a VPL activity

As an example, Figure 83 shows the description of a new exercise in the VPL.

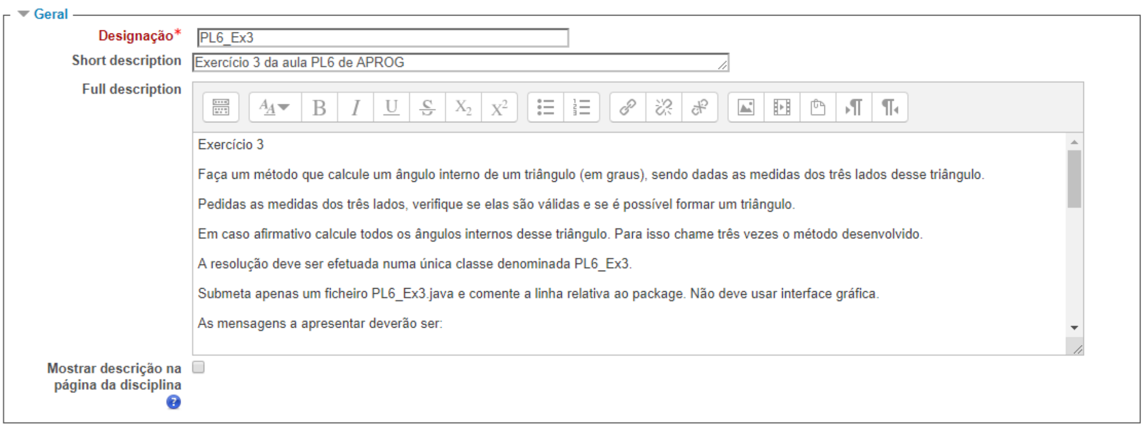


Figure 83 - Description of a new VPL exercise / activity

Although, in this case study the VPL has not been effectively used for the evaluation process, it is possible to use it for this purpose, in which case it is necessary to make the associated settings, particularly with regard to scale. It is also possible to define the grade for approval, as well as the maximum number of possible submissions without penalty and the reduction of the grade for each additional submission (Figure 84).

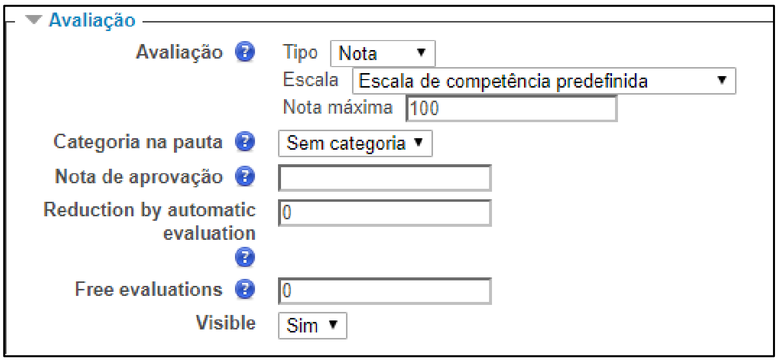


Figure 84 - Setting evaluation parameters

The grade will be obtained according to the scale and successful tests.

After the complete setting up of the activity it was necessary to configure the technical and functional features (Figure 85).

One of the most important aspects, which has a direct relationship with the pedagogical perspective of the process, was the definition of the test cases, which are crucial for the success of the process, and should therefore be exhaustive and numerous, covering all possible situations.

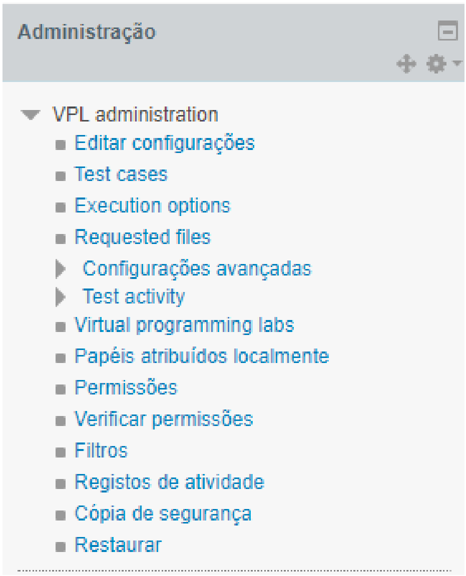


Figure 85 - VPL administration menu

The test cases are specified in the file *vpl_evaluate.cases* (Wanhenheim, Martina, Cancian, & Dovichi, 2015), and it is necessary, in each test, to indicate the test name (case), which input data will be tested (input) and which output is expected (output), as shown in the example of Figure 86.

vpl_evaluate.cases

```
case = Teste 1
input =
11.0
11.0
15.5564

output=
output=90.00
45.000
45.000

```

Figure 86 - Example of a test case definition

The grade of each test is determined automatically, being obtained by the ratio between the maximum value of the scale and the number of test cases. There may, however, be test cases considered more important than others, and the grade should be attributed to each one according to the complexity and dimension of what is to be tested. It is therefore possible to define a grade penalty (as a percentage of the maximum value of the scale), and after the test output the grade reduction = X% instruction should be added, X representing the value to be specified.

In the settings it is still necessary to define the execution options that include the definition of the actions allowed to the students, regarding execution, debugging or evaluation of their work (Figure 87).

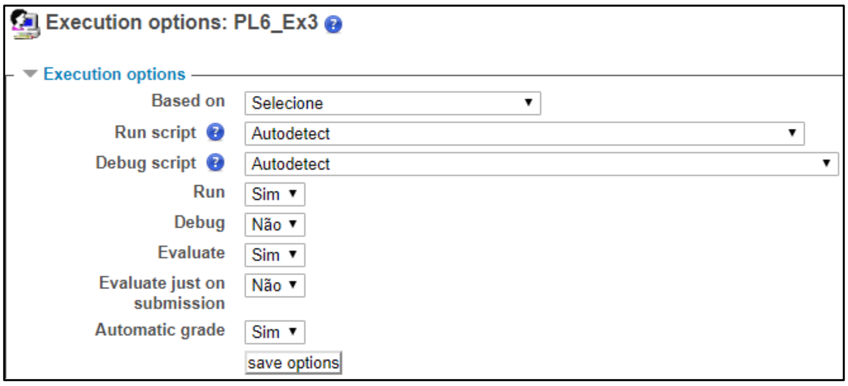


Figure 87 - Execution options

In addition to the test case file (*vpl_evaluate. cases*) the VPL considers three more files for specific execution settings (*vpl_run.sh*), debugging (*vpl_debug.sh*) and evaluation (*vpl_evaluate.sh*). In this experiment, the focus being on the pedagogical process and on *feedback*, the additional possibilities regarding debugging and evaluation were not considered relevant and were not explored, so only the files *vpl_run.sh* and *vpl_evaluate. cases* were used. Even though the VPL identifies the type of file by extension and invokes the corresponding compiler, it was intended that the compilation be done only in Java, since it was the language used and the only one allowed in APROG.

Thus, it was chosen to explicitly indicate the name of the file to compile and execute in the script of execution (Figure 88).

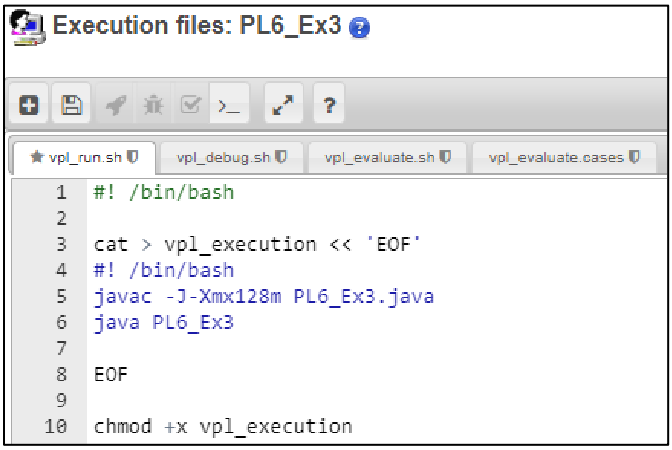


Figure 88 - File*vpl_run.sh*

After the creation of the six chosen activities, all the exercises were previously tested, with the necessary corrections and changes for proper operation. At the end of this stage each exercise was again tested with a specific user with a unique student profile in order to ensure all the conditions of execution.

The study took place between the school years 2017/2018 and 2019/2020. At the end of the first period an assessment was made, and a plan was elaborated that led to the reengineering of the process, with significant changes.

The following sections aim to present a description of the implementation of activities in the VPL in each of the school years in which the experiment took place, with an explanation of the specificities and the changes that have been introduced.

5.4.3 2017/2018 school year

At the beginning of the 2017/2018 school year, 314 students were enrolled in APROG, divided into 17 classes, of which eight were pilot classes dedicated to the use of VPL.

In these eight classes, there were 161 students enrolled of whom only 135 actually attended lessons. All 161 students were automatically enrolled in APROG-VPL 2017/2018 and organized into groups specifically created for this purpose and corresponding to the PL class to which the student belonged.

In this school year, the APROG organization corresponded to three sprints, in the eduScrum (Delhij, van Solingen, & Wijnands, 2015) model, having been changed from the previous school year where four sprints had been considered, instead. In this new distribution, the sprints were organized according to Table 18.

Table 18 - Sprint organization

Sprint	Number of weeks	Content
1	4	Algorithmics. Pseudocode. Flow charts. Tracing.
2	4	Implementation of algorithms in Java, modular decomposition, and one- and two-dimensional indexed data structures.
3	4	Development of an application integrating all the contents taught.

As mentioned before, for the use of VPL an area was created in Moodle, initially called APROG-VPL which was later renamed APROG-VPL 2017/2018.

The use of VPL started in the sixth week of lessons (second week of sprint 2), for the reasons previously mentioned related to the gradual introduction of new contexts. For each of the remaining three weeks of this sprint two exercises were chosen and adapted for submission to the VPL, covering the generality of the contents taught.

In Figure 89 it is possible to see the student's view of Moodle of the six chosen exercises.

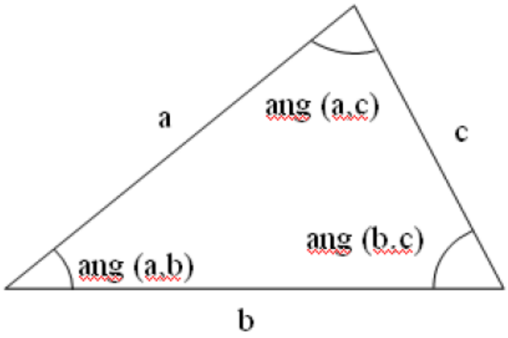
Figure 89 - Exercises in the VPL - 2017/2018

As an example, the original statement of exercise 3 of class PL6 is presented in Figure 90.

Exercício 3 (**)

a) Faça um método que calcule um ângulo interno de um triângulo sendo dadas as medidas dos três lados desse triângulo. O valor do ângulo deve estar em graus.

b) Sendo dadas as medidas de três lados, verifique se as medidas são válidas e se é possível formar um triângulo. Em caso afirmativo calcule todos os ângulos internos desse triângulo. Para isso chame três vezes o método desenvolvido na alínea anterior.



Ângulo	Fórmula
ang (a,b)	$arc\cos\left(\frac{a^2+b^2-c^2}{2ab}\right)$
ang (a,c)	$arc\cos\left(\frac{a^2+c^2-b^2}{2ac}\right)$
ang (b,c)	$arc\cos\left(\frac{b^2+c^2-a^2}{2bc}\right)$

Figure 90 - Exercise 3 of PL6 class - 2017/2018

As already mentioned, the functioning of the system is based on the analysis and evaluation of test cases, the result being determined according to the similitude of Strings, relative to the result obtained and the expected result. Hence, this is a crucial aspect of system operation, so examples of *inputs* and their expected outputs should always be specified.

For the exercises in question, the necessary adaptations were made to the assignment in order to reduce, as far as possible, any ambiguities, by mentioning some particularities to be considered when submitting to the VPL and presenting examples of the input data format and the expected results, as can be seen in Figure 91.

PL6_Ex3

Due date: Sexta, 15 de Dezembro de 2017 às 23:55
Maximum number of files: 1
Type of work: Individual work
Exercício 3

Faça um método que calcule um ângulo interno de um triângulo (em graus), sendo dadas as medidas dos três lados desse triângulo.

Pedidas as medidas dos três lados, verifique se elas são válidas e se é possível formar um triângulo.

Em caso afirmativo calcule todos os ângulos internos desse triângulo. Para isso chame três vezes o método desenvolvido.

A resolução deve ser efetuada numa única classe denominada PL6_Ex3.

Submeta apenas um ficheiro PL6_Ex3.java e comente a linha relativa ao package. Não deve usar interface gráfica.

As mensagens a apresentar deverão ser:

Para pedir os dados:

Lado a?

Lado b?

Lado c?

Para apresentar resultados:

Angulo(a,b)= 60,000º

Angulo(b,c)= 60,000º

Angulo(c,a)= 60,000º

No caso de o triângulo não ser possível, a mensagem deverá ser:

Triangulo impossivel

Figure 91 - Exercise 3 of PL6 class in the VPL - 2017/2018

In order to smooth the process of students' initiation into the use of the VPL, a brief explanation and a face-to-face demonstration in each of the classes where the VPL would be used was carried out at the beginning of the first class of the sixth week by the author of the study.

After this stage, the students would work out the solutions for the exercises in the IDE and, for the mentioned exercises, they would submit them to the VPL.

Initially there were some difficulties in the execution and validation of exercises in the VPL, mainly due to non-compliance with the specific indications for submission.

During the experiment, some difficulties were observed due to the novelty of the process. However, the evolution of the students' ability to carry out the activities as well as the enthusiasm with which they did so was also remarkable.

Although it was explicitly stated that only two exercises per week would be submitted in the VPL, some students asked the professors involved in the study how they could send more exercises to check whether they were correct. It was explained to them that we did not have, at that stage, any more exercises prepared for the VPL and that this preparation was demanding and time consuming so that there were no conditions to do so in due time.

As previously mentioned, the eduScrum methodology was used, promoting collaborative activities with students working in pairs.

In this way, of the 135 students who could potentially submit the exercises, many did so as a group, which substantially reduced the number of submissions. Since this is a voluntary process, several students did not join for reasons, expressed verbally, related to lack of time and/or interest in the process.

Table 19 shows the data on the number of submissions made.

Table 19 - VPL submissions in 2017/2018

Exercises	PL6		PL7		PL8	
	Ex3	Ex5	Ex2	Ex4	Ex2	Ex3
Submissions	66	41	47	40	30	21

In the planning of this school year three individual evaluation moments were defined, at the end of each of the three sprints. Due to the lack of logistic conditions to perform computer evaluations, these were performed on paper.

To assess the potential of the VPL in a simulated mid-term perspective, after the second moment of face-to-face assessment, which took place at the end of sprint 2, students were invited to submit their performance to the VPL.

As the students were already familiar with its use, it was decided, as mentioned and on an experimental and merely demonstrative basis, to use the VPL so that each student could test, on the computer, the resolution they had made on paper.

Therefore, after the evaluation and classification of the resolutions by the teacher, each student was given their resolution and challenged to test it in the VPL (Figure 92).

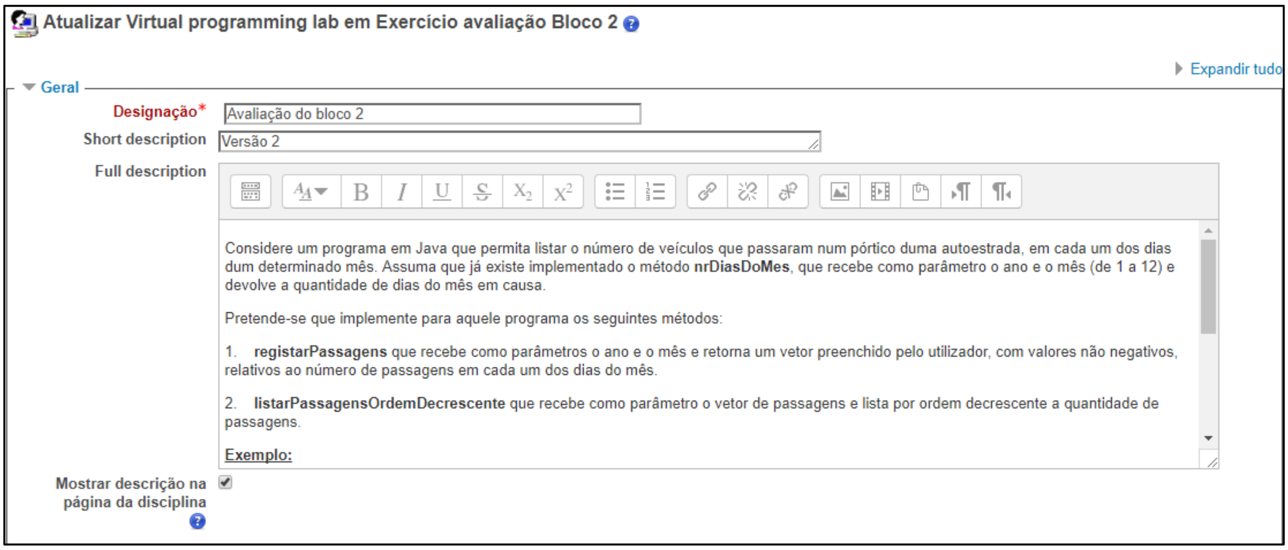


Figure 92 - Example of evaluation exercise in the VPL

This process aimed to carry out a test in order to evaluate the possibility that, if and when the necessary logistic conditions are met, the VPL can be used for evaluation purposes. It also served for students to test their code and analyze its resolution and the grade given by the teacher, to clarify the process of evaluation and *feedback* to reinforce learning.

At the end of the 2017/2018 edition of APROG a review was made and an evaluation of the use of the VPL was carried out. Student surveys (Annex F) were carried out, the results of which were not subject to in-depth analysis, but it can be said that in general terms the opinions were strongly in favour of the use of VPL.

5.4.4 Process adjustments and re-engineering

Despite the generally positive opinion, some aspects were pointed out as potential targets for improvement, in order to minimise the probability of errors and difficulties occurring and, on the other hand, to improve feedback to the student, increasing the contribution of the system to the improvement of its teaching-learning process.

Thus, a number of changes have been promoted to overcome some technical difficulties, but mainly for pedagogical reasons.

5.4.4.1 Technical Issues

One of the most frequent errors was the discrepancy between the class name defined in the code (and its file name) and that specified in the assignment (Figure 93).

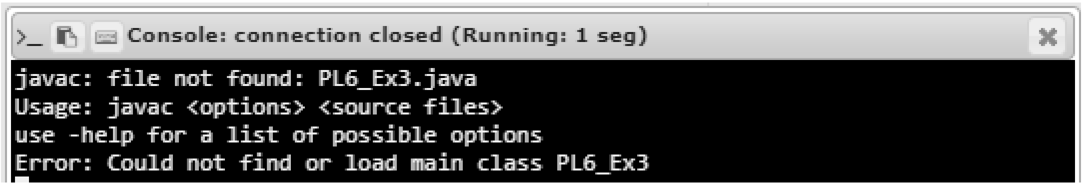


Figure 93 - Error due to missing file to be compiled

Attention was paid to this aspect and a solution was developed which consisted of searching the java file where the *main* class was in the submission.

With this change, submissions could now be made without needing to use a specific name in the file for a successful submission. This has proved to be a very effective measure in significantly reducing the problems encountered in submissions.

As mentioned, due to technical limitations of the Linux server available, it was not possible to use components from graphical environments. Thus, this verification is now carried out upstream, the user being informed of this impossibility if the import of Java Swing[106] or Java FX[107] libraries is identified in the code.

5.4.4.2 Pedagogical aspects and implementation of verification mechanisms

In addition to the issues mentioned, needs related to the pedagogical process and in particular to feedback to the student were identified. The VPL analyses the result only, but in our case, for the pedagogical process, it was fundamental to analyse the codification process as well, i.e. the fact that the student's work generates a result equal to the expected did not guarantee that all requirements had been met or that the codification developed was the most appropriate approach, in terms of solution and its implementation.

To refine the analysis process, some aspects have been identified which should be verifiable.

[106] Java High Level Graphics API
[107] Platform multimedia developed by Sun Microsystems

Based on the PL classes and the opinions collected from the colleagues who taught them, it was found that the students usually show difficulties in the modular decomposition of the code. The use of methods, the passage of parameters and the return of results are, in a first approach, concepts that are difficult to understand and incorrectly applied.

To get around these difficulties, students wrongly tend to avoid modular decomposition by executing all the instructions in one method. This is a recurring situation and only detectable by analysis of the code and often the coded exercise is poorly coded but results in the desired output.

Taking into account that good programming practices are a very important aspect and that they should be induced from the beginning of learning (Horstmann, 2013) it was decided that a mechanism for verifying the use of methods should be implemented in several areas.

One of the aims was to check whether a particular method was declared and invoked, the name of which would be explicitly defined in the problem statement. Another check would be determining the quantity of methods used. And in a third, it was intended to determine the length (number of lines) of the method, since each method should serve only one functionality, so if it is too long (Badri, Badri, & William, 2016) it may indicate that it has several functionalities and/or be inefficient.

Also, with regard to modular decomposition, associated with the re-use of code, it is often the case that students repeat instructions resulting in confusing, inefficient and overly extensive code. As previously mentioned, the proposed exercises are previously solved by the teachers, it being possible to estimate a reasonable maximum number of lines of code and, by analysing the student's resolution, to verify if their code meets this criterion. The possibility of defining the maximum number of lines and the automatic accounting of the number of lines used has been implemented. The coding of this functionality has required some refinement as only the instruction lines should be considered, thus excluding space lines between instructions as well as all lines relating to comments.

Also, in relation to good programming practices, the use of constants should be considered when there are fixed values in the code, something that many students did not respect either. Since the only way to verify compliance with this practice was by direct analysis of the code, a mechanism has been devised which quantifies the "final" existing statements in the code and, if it finds them, checks whether the name of the constant has been defined in capital letters, as advocated by the convention[108].

As the compilation and execution of the VPL is ensured by an initialization file called *vpl_run.sh*, some of these checks were implemented and tested gradually and individually by the inclusion of instructions in bash[109] in that file. However, in creating a new VPL activity, the use of the various verification functionalities implied copying it into the new *vpl_run.sh* activity, rewriting and/or commenting lines to perform only the desired checks.

If a change, improvement or correction of the *bug* is made, that change should be implemented in all the activities already created, editing and changing the *vpl_run.sh* file of each one. This model was complex to maintain, which implied great commitment to updating and replicating the mechanisms in each new activity created.

Apart from this, the specific development of new checks proved to be very difficult to implement in bash so a PHP solution was tried, since the language was already installed on the server used and had the necessary functions for Java code analysis. Thus, a file (*vpl_php.sh*) was created where the most complex code analysis features were included.

[108] https://www.oracle.com/technetwork/java/codeconventions-150003.pdf
[109] Shell developed for the GNU project, standard in distributions Linux

In this model, the checks are now centralised in a new file (*vpl_run_include.sh*) with Linux bash functions for validation. This file was hosted in a bitbucket[110] repository, and therefore is always updated and available for each VPL activity. However, after some tests there were some constraints in accessing the repository, either because of momentary connection difficulties or because of excessive slowness in the execution of the processes.

A solution with two repositories was adopted, the first being to summon a repository at a local address in the ISD, and if access was not possible or a file was not found, only then would an attempt be made to access the bitbucket repository. This has created redundancy in access, increasing the speed and reliability of the process, while requiring any updating to be carried out in both repositories.

5.4.4.3 Implementation of solution with parameterization and shared resources

Due to the difficulties mentioned in maintaining the verification mechanisms, a solution was designed to replace the commands for compiling and executing *vpl_run.sh* with a specific set of commands that would allow the analysis of the code before the compilation and execution attempt.

As the validation process was centralized, the system had to be able to test the desired functionalities only, and a configuration structure common to all exercises had to be defined. This approach allowed, for each exercise, to define and parameterize what was to be evaluated, with the following points being considered:

- Check the concordance between the file name and the class name;
- It warns of the impossibility of using Java Swing or Java FX;
- Check for the existence of a method with a specific name;
- Compare the overall number of lines of code with a predefined maximum value;
- Determine the quantity of methods used and compare it with pre-defined minimum and maximum values;
- Determine the quantity of constants declared and compare it with pre-defined minimum and maximum values;
- Determine the number of lines of a method with a specific name and compare it to pre-defined minimum and maximum values.

All these specifications are optional, the teacher who creates the activity being responsible for defining which one or more to use and whether the verification should lead to an error or a warning. In any case the user receives a message, but if it is defined as an error, execution is inhibited.

For the implementation of these checks three *arrays* were included in the file *vpl_run.sh*, the first one ("validations_menu") being the reference of the functionality to be tested according to the one presented in Table 20.

[110] https://bitbucket.org/

Table 20 - Configuration parameter references

Reference	Functionality
cls	Class / file name
gui	Graphic interface
fnc	Specific method
lin	Total number of lines
met	Quantity of methods
cns	Constants
met_nam1	Lines of a specific method

The decision to test each functionality is up to the person who defines the activity, according to the specificities of each exercise and what is to be verified.

This choice is made when filling in the array "validations_todo" by indicating in each position "1" or "0", depending on whether you want to test the respective functionality or not.

The last array ("validations_error") allows to classify the test as an error or merely as a warning with "1" or "0" respectively.

Thus, each functionality will or will not be tested as defined in "validations_todo". If the functionality is tested and a non-conformity is found, the corresponding position in "validations_error" will be analysed.

If in this array the value "1" is found, it will be considered an error and the execution is interrupted, otherwise only a warning message is generated, and the process continues.

For each of the checks, messages are defined for each VPL activity.

For example, in Figure 94 the limits are defined (in vpl_run.sh) as well as the message to be added if the total number of lines in the code exceeds the defined maximum value.

```
v_lin_max=40
v_lin_err_msg="O número de linhas supera o número de linhas esperado.
Reescreva a sua aplicação de modo a ter, no máximo, $v_lin_max linhas.
Obrigado."
```

Figure 94 - Definition of maximum number of lines and their error message

In this example the maximum number of lines was set at 40. Thus, if this limit was exceeded, the message on Figure 95 would be generated.

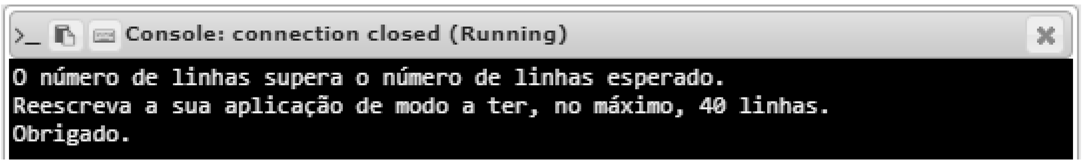


Figure 95 - Error message regarding maximum of lines

This way the file *vpl_run.sh* is used as the basis of the process and now incorporates a set of parameters that allow the specification of the checks for each exercise.

5.4.4.4 *Cloud solution refinement*

The solution developed so far met the general objectives of the validations intended to serve pedagogical purposes globally, although still presenting some constraints.

Although the checks were centralized, the parameterization and messages would have to be defined locally for each activity.

It was also found that although the process was centralized it would be convenient to have a stable and functional version while making corrections or testing new features.

In view of these issues, new developments have been made which have included the creation of new files: *vpl_prm.sh* and *vpl.version* and the redefinition of the *vpl_run.sh* file structure.

In this much simpler model, *vpl_run.sh* now contains the following sections:

- static statement;
- version management and download files;
- parameter replacement;
- validation, compilation and implementation.

In *vpl_prm.sh*, standard parameters were defined, usable for all VPL activities. Thus, there is now a basic parameterization for defining limits and messages. However, if in each activity there is the definition of any of the parameters, this is the value that becomes considered for the process overlapping the standard parameterization. In this way, it is only necessary to define locally whatever is meant to be different from what is included in the global parameterization.

The *vpl.version* file only has the definition of the versions of each of the files (*vpl_prm.sh*, *vpl_php.sh* and *vpl_run_include.sh*) that is intended to be used. This allows for stable versions while others are in production without disturbing the normal functioning of the system.

Table 21 shows the files used in the process and the functions provided by each one.

Table 21 – Files and their functions

File	Functions
vpl_run.sh	Version control and repository downloads Defining specific parameters
vpl_prm.sh	Standard parameters and validations Standard error messages
vpl_run_include.sh	File cleaning Bash validations
vpl_php.sh	Complex validations
vpl.version	Version definition

In this model, the operation of the solution can be represented as shown in Figure 96.

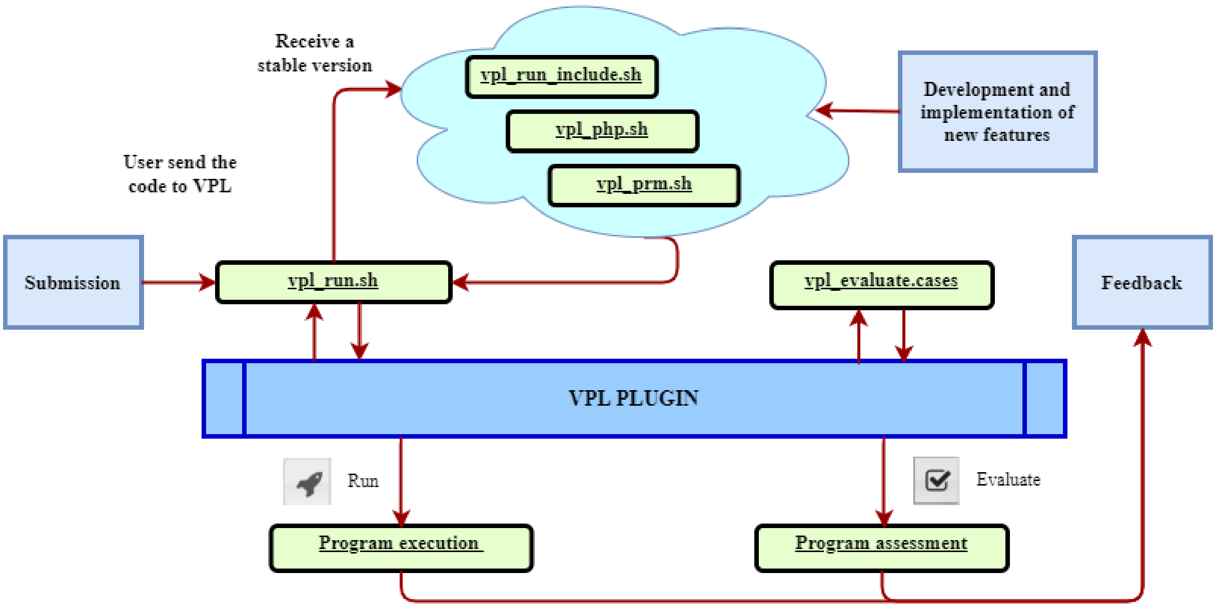


Figure 96 - Solution cloud with version control

This new strategy simplifies the validation code management process, facilitating the inclusion of new resources without disrupting the functioning of existing exercises. It allows debugging new versions by creating a local *vpl.version* file that indicates the desired version.

When the process starts, the files of the most current stable version (or the desired version, if there is *vpl.version* locally) are downloaded from the repository (internal to DEI or bitbucket) and executed locally.

Each attempt to execute or validate the VPL activity (within the same session) is checked to see if download has already been carried out and, if so, the local files used. This check prevents the consecutive downloading of files by significantly reducing the process' execution time.

5.4.5 School year 2018/2019

In the second year of the experiment (2018/2019), the structure, mode of operation and content organisation of the APROG CU remained the same as in the previous year. In this edition of APROG, 307 students were enrolled, placed in 17 classes, 13 of which were taught by teachers who participated in the experience of using the VPL.

In the universe of these classes there were 235 students enrolled, of which only 217 would actually attend the lessons. All 235 students were enrolled in APROG-VPL 2018/2019 and grouped according to their PL class.

To carry out the experiment, the process of creating a new area in Moodle was repeated for the release of a new CU called APROG-VPL 2018/2019. This new CU was created from the import of the APROG-VPL 2017/2018 area, in order to incorporate all the content of this area.

On this basis, the necessary changes have been made to the operation of the system with version control as described in section 5.4.4.4. An initial section was also added with the background of the study and general indications on the specificities to be observed in the elaboration of the resolution (Figure 97).

Virtual Programming Lab

Virtual
Programming Lab

Caro(a) aluno(a),

Tem acesso a esta funcionalidade devido a estar inscrito na Unidade Curricular (UC) de Algoritmia e Programação (APROG) e pertencer a uma das turmas onde se irá efetuar uma experiência pedagógica.

Pretende-se realizar um estudo sobre a integração de Virtual Programming Lab (VPL) no processo de ensino/aprendizagem de programação na UC de APROG.

Para tal solicita-se a sua colaboração que consistirá em testar alguns dos exercícios das fichas das aulas PL, no VPL, seguindo as instruções do seu professor das aulas PL.

Para a submissão dos seus exercícios deve ter em atenção o seguinte:

- A solução deve ser desenvolvida numa única classe.
- Deve executar **exclusivamente** em modo texto (sem recursos swing, javafx ou outros de ambiente gráfico).
- Para testar a sua solução foi estabelecido um plano de testes com um formato específico de output. Para que os testes possam ser realizados, com sucesso, deverá cumprir fielmente o estipulado no enunciado.

Pode encontrar mais informação em http://vpl.dis.ulpgc.es/

Figure 97 - Initial section APROG-VPL 2018/2019

In order to foster familiarity with the process of using the VPL and, in particular, of code submission, a demonstration activity of the functioning of the VPL was designed and implemented (Figure 98).

Exercício demonstrativo do funcionamento do VPL

Exercício demonstrativo

Desenvolva um programa em Java que calcule e apresente todos os divisores de um número inteiro positivo, definido pelo utilizador, e que sejam múltiplos de 5.

Caso não exista nenhum número nestas condições deverá ser enviada para o ecrã a mensagem: **sem resultados**

Deve implementar o método *lerInteiroPositivo* que permita ler um número inteiro e validar se ele é positivo.

A solução deve ser desenvolvida numa única classe e exclusivamente em modo texto.

Exemplo:

Entrada

Qual o número?

465

Resultados

5

15

155

465

VPL_Exerc_Demonstrativo.java

Este ficheiro serve de base para testar o exercício demonstrativo.

Faça as alterações necessárias para que funcione do modo pretendido.

Figure 98 - Exercise demonstrating the functioning of the VPL

In the activity a file with Java code was provided, which had some logical inaccuracies and did not meet all the requirements of the problem. If the student used the code provided without any changes, they would get some errors that they would have to correct.

After this introduction exercise, the same exercises were made available in Moodle as in the previous year, although they were fully revised. Changes were made in the test cases, increasing the number of tests on each activity and correcting some inaccuracies identified in the previous year. As the comparison of *Strings* is the basis for checking the functioning of the VPL, the formatting required in *output* in each of the exercises has been greatly simplified in order to minimise the constraints of the previous year and avoid any "false negatives".

These changes involved re-writing the wording of the exercises to make the new specifications of output more explicit and adapting the examples given, and changes were made to the text to avoid any ambiguity. The changes made were replicated in the APROG exercise list sheets, replacing the existing versions. This action aimed to facilitate the articulation between the development process in FDI and submission to the VPL, minimizing the need for changes imposed by the VPL restrictions since, at the outset, these would already be contemplated.

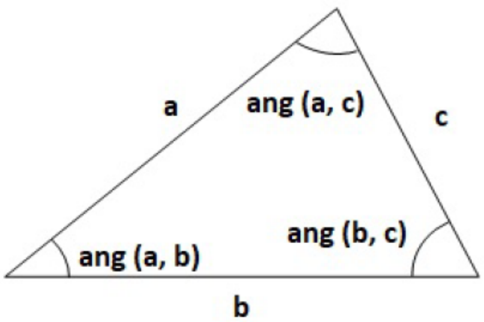
As an example, in Figure 99 the APROG Factsheets exercise 3 of PL6 class, 2018/2019, shows the changes from the previous version (Figure 90).

Exercício 3 (**)

a) Faça um método (**calcAng**) que calcule um ângulo interno de um triângulo (em graus), sendo dadas as medidas dos três lados desse triângulo.

b) Pedidas as medidas de três lados, verifique se elas são válidas e se é possível formar um triângulo. Em caso afirmativo calcule todos os ângulos internos desse triângulo. Para isso chame três vezes o método desenvolvido na alínea anterior.

Nota: A solução deve ser desenvolvida exclusivamente em modo texto. Nos resultados deverão ser apresentados **apenas** os valores dos ângulos arredondados às unidades, sem qualquer texto ou outra informação. No caso de o triângulo não ser possível, a mensagem deverá ser: **impossivel**



Ângulo	Fórmula
$ang\ (a,b)$	$\arccos\left(\frac{a^2+b^2-c^2}{2ab}\right)$
$ang\ (a,c)$	$\arccos\left(\frac{a^2+c^2-b^2}{2ac}\right)$
$ang\ (b,c)$	$\arccos\left(\frac{b^2+c^2-a^2}{2bc}\right)$

Figure 99 - Exercise 3 of PL6 class - 2018/2019

As can be seen in Figure 100, the statements in the APROG sheets and in APROG-VPL 2018/2019 are now similar.

APROG - Aula prática-laboratorial 6

Pretende-se que estas atividades permitam aos alunos a submissão do código Java dos exercícios números 3 e 5 da ficha PL6.

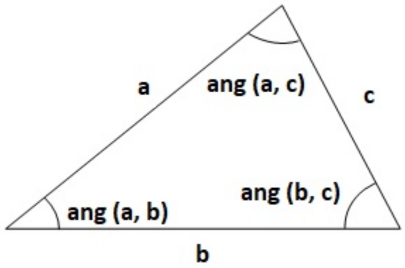
O objetivo é que validem o seu código de forma automática, obtendo informação sobre a validade da solução implementada, verificada pela execução de testes automáticos previamente definidos.

PL6 - Exercício 3

Faça um método (**calcAng**) que calcule um ângulo interno de um triângulo (em graus), sendo dadas as medidas dos três lados desse triângulo.

Pedidas as medidas dos três lados, verifique se elas são válidas e se é possível formar um triângulo.

Em caso afirmativo calcule todos os ângulos internos desse triângulo. Para isso chame três vezes o método desenvolvido anteriormente.



Ângulo	Fórmula
$ang\ (a,b)$	$arc\ \cos\left(\frac{a^2+b^2-c^2}{2ab}\right)$
$ang\ (a,c)$	$arc\ \cos\left(\frac{a^2+c^2-b^2}{2ac}\right)$
$ang\ (b,c)$	$arc\ \cos\left(\frac{b^2+c^2-a^2}{2bc}\right)$

A solução deve ser desenvolvida numa única classe e exclusivamente em modo texto.

Nos resultados deverão ser apresentados **apenas** os valores dos ângulos arredondados às unidades, sem qualquer texto ou outra informação, pela ordem (a,b), (a,c) e (b,c).

No caso de não ser possível formar um triângulo, deverá apresentar a mensagem: **impossivel**

Exemplos:

Entrada de dados	Resultados	Entrada de dados	Resultados
Lado a: 5.0	**80**	**Lado a: 5.0**	
Lado b: 6.5	**59**	**Lado b: -6.5**	
Lado c: 7.5	**41**	**Lado c: 7.0**	**impossivel**

Figure 100 - Exercise 3 of PL6 class - 2018/2019

This being the second year of use of the VPL, the author of the study and the teachers contributed to it a more in-depth knowledge of the technical and pedagogical issues involved in the process. It was possible to avoid several problems and to anticipate and solve others much more simply and quickly.

As the process continues to be one of voluntary enrolment by students, the teaching team has been better prepared to motivate them to enroll by explaining the process earlier on and in more detail and on a more sustained basis. Thus, of the 217 students who attended APROG in the classes where the VPL was used, 196 submitted at least one of the suggested exercises.

Table 22 shows the number of submissions, for each exercise, in absolute value and as a percentage of the total number of students who could potentially use the VPL.

Table 22 - 2018/2019 VPL submissions

Exercises	PL6		PL7		PL8	
	Ex3	Ex5	Ex2	Ex4	Ex2	Ex3
Submissions	196	191	183	181	191	177
	90,3%	88,0%	84,3%	83,4%	88,0%	81,6%

The overall average percentage of submissions (considering all exercises) was 85.9%, representing a significant increase in participation compared to the previous year where it had been of 30.2%.

Overall, the experiment went very well, without relevant technical problems or difficulties for most students.

It was found, however, that some pupils had abnormally high numbers of submissions (Figure 101).

Figure 101 - Example of the number of submissions for an activity in the VPL-2018/2019

When asked why this was so, it was found that in some cases students did not reflect on why the solution was incorrect, using the VPL to test their solution in a trial-error strategy until the desired result was obtained. This is obviously a misuse and contrary to the intended objectives.

In view of this, the decision was taken to re-explain the process and its objectives.

Although the VPL results were not used for grading purposes, students endeavoured to have their programmes pass all the tests in order to achieve the maximum grade. Thus, for the exercises that were still to be launched (the two in PL8), a feature of the VPL was used to limit the number of submissions with no penalty in classification. It was decided to limit this number to 4, with the grade being reduced by 5 values for each additional submission (Figure 102), to a maximum of 100.

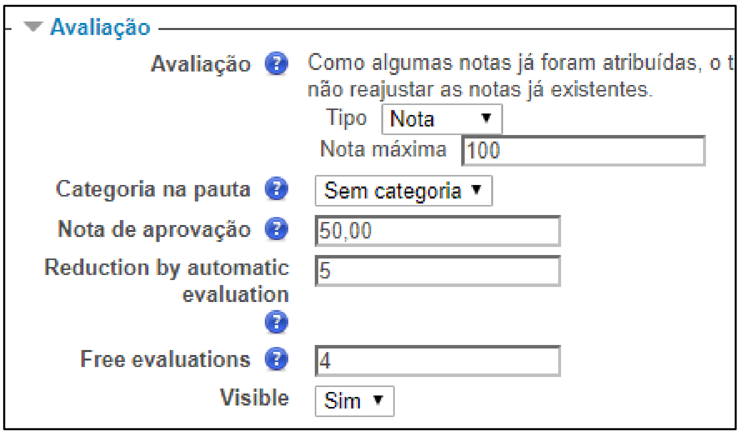


Figure 102 - Setting limits on the number of submissions

Following these measures there was a drastic reduction in the number of attempts at submission.

At the end of the process an automatic evaluation simulation experiment was also planned. This experiment aimed to test the conditions for carrying out automatic assessments using the VPL as well as the safety and reliability of the process.

To this end some students from one of the PL classes taught by the author of this work and in which the VPL had been used were approached. In this meeting the context and objectives of the experience were explained, and eight students showed their willingness to join in.

It was then necessary to find a date compatible with everyone's availability and after the analysis of various alternatives, the simulation was scheduled for 14 January 2019.

For the simulation, a specific exercise was prepared (Figure 103) to which only stakeholders were given access.

Options directly related to an evaluation context were used in the exercise settings:

- the use of a file with a specific name was mandatory;
- highest grade = 100;
- number of submissions without penalty = 2;
- penalty for each additional assessment = 5;
- use of password;
- inability to upload files;
- inhibition of editing functionalities, namely copying and pasting content of external origin.

The following figure shows the exercise created and, in addition to the assignment, the configurations of the activity can be observed.

Exercício de avaliação

Avaliable from: Segunda, 14 de Janeiro de 2019 às 14:45
Due date: Segunda, 14 de Janeiro de 2019 às 16:00
Requested files: VPL_avaliacao.java (Download)
Type of work: Individual work
Definições de notas: Nota máxima: 100
Reduction by automatic evaluation: 5 **Free evaluations**: 2
Senha: Sim
Dissable external file upload, paste and drop external content: Sim
Run: Sim. **Evaluate**: Sim
Automatic grade: Sim.

Considere um programa em Java que permita listar nomes alfabeticamente até a um máximo de 5 nomes.

Para tal deve implementar as seguintes funcionalidades:

1) **preencherArrayNomes** que recebe como argumentos um array vazio e o preenche com nomes até ser digitado "FIM", devolvendo a quantidade de nomes existentes no array.

2) **listarNomes** que recebe como argumentos um array de Strings e lista os elementos do array (um por linha) por ordem alfabética.

A solução deve ser desenvolvida numa única classe (VPL_avaliacao) e exclusivamente em modo texto, efetuando decomposição modular, respeitando as convenções do Java e outras boas práticas de programação.

Exemplo:

Dados de entrada: Pedro Costa, Ana Silva, Rosa Lopes, FIM

Saída:

Ana Silva

Pedro Costa

Rosa Lopes

Figure 103 – Automatic evaluation simulation exercise

This simulation contained some technical constraints, namely regarding network access control, so that it was limited to the Moodle server and the jail server required to perform the VPL activities, such restriction being made by changing the network settings.

In order to enable classes or other activities to function normally, it was not possible to make changes to the existing network, so a specific IP network was created for exclusive access to Moodle and VPL activities. This new network has become active and available for possible future use.

To ensure the technical conditions for carrying out the experiment, the following actions were thus carried out:

- creation of a specific IP network;
- creation of a new pool of IP addresses for the new network;
- setting rules in the *firewall* for the new network, conditioning access to the Moodle server, the jail server and the internal DEI DNS servers;
- creation of a new local area network (LAN) associated to the new network;
- configuration of the new LAN in the various DEI network equipment so that it is accessible in the classrooms.

When the test was conducted in Moodle, the equipment in the room where it took place would have to receive the network addresses for that purpose. Thus, it was necessary to change the VLAN on the *switch* of the room to the VLAN associated with the specific network.

The activity was carried out on the scheduled date, with seven of the eight students who initially made themselves available attending. The action started with the credentialing of the students in Moodle with the activity only available after the entry of a password, which was designed and made available to everyone simultaneously.

In the end, students were asked verbally about the experience, the general opinion being very favourable. Five of the students obtained the highest score, the remaining 80 and 20 respectively. In the case of the last student, the classification obtained was due to the high number of attempts until the expected result was obtained.

This is an interesting potential of the VPL, but whose mass implementation is not feasible in the DEI, with the current logistical and technical conditions. Of course, it would still have to be further explored and tested, but if it could be implemented, it could add great value to the evaluation process, with automatic and immediate grading.

In any case, the technical conditions for future experiments and, possibly, the use of VPL in a real evaluation context have been created.

5.4.6 School year 2019/2020

In the initial planning of this work no activities were planned for the school year 2019/2020. However, following the experience acquired in the two previous years and given the good acceptance by the students and the fact that the author of this study continued to be part of the APROG teaching team, the possibility of using the VPL in this edition of the CU was once again considered.

A new APROG RUC for the school year 2019/2020 was designated and was contacted in order to maintain the use of VPL in the model previously used. In that meeting, the RUC reported being aware of the study carried out in previous years and had planned to extend the use of a code evaluation tool, in a systematic way, to all exercises and to all students.

However, since he was not familiar with the functioning of the VPL and had previously used Mooshak in several programming competitions, he wanted to opt for the use of Mooshak.

However, he agreed to the use of VPL after Mooshak submissions and in a complementary manner.

The new RUC of APROG introduced some changes to the operation of the CU, although the course content and the same division into three blocks were maintained.

The main change was made to the operation of the second block dedicated to coding in Java, in particular with the use of Mooshak. As this is a tool originally designed for the management of programming tenders, as mentioned in section 4.2.3, three tenders were created: “Java – essential”, “Java – modularization” e “Java. – arrays”.

This was also the organisation used in the exercise sheets provided in Moodle. The sheets were drawn up on the basis of the existing exercises in previous years' sheets, which were revised and adapted to the operation of Mooshak, with new exercises being added. For each exercise, tests were defined and each one was coded to test it on Mooshak.

Another change concerned the non-use of graphic components (swing) because they are not relevant for this learning stage and because their use is not possible in Mooshak.

For the VPL, the same exercises were used as in the previous year, with the necessary adaptations to the wording prepared for Mooshak, with the exception of the last two, relating to the PL8 sheet, which were not used because they had been removed from the sheets in this edition of APROG.

Table 23 shows the correspondences between the names of the common exercises for the two school years.

Table 23 - Naming of VPL exercises in different years

	Exercises			
2018/2019	PL6_Ex3	PL6_Ex5	PL7_Ex2	PL7_Ex4
2019/2020	Modularização_C	Modularização_E	Arrays_E	Arrays_H

As in previous years, a new import area was created in Moodle from the APROG-VPL 2018/2019 area and was named APROG-VPL 2019/2020.

The necessary changes were made to the new context, with an initial section contextualising the use of VPL and a call for the participation of pupils (Figure 104).

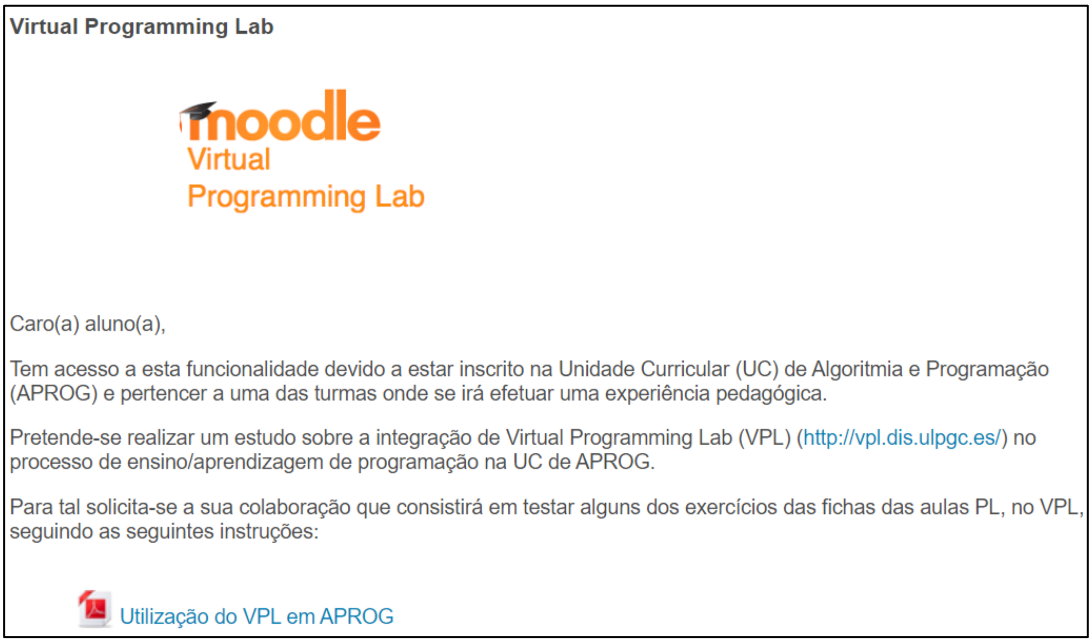


Figure 104 - Initial section of Moodle APROG-VPL 2019/2020

As the VPL was used for the Mooshak exercise test, a text explaining the use of the VPL in 2019/2020 (Annex K) was prepared and made available in the aforementioned section of Moodle.

For the registration of students in APROG-VPL 2019/2020, the same model was used as in previous years, with automatic registration organised by groups corresponding to the PL classes. The use of Mooshak preceded the use of VPL and the students who made submissions to Mooshak were identified and only those were registered in APROG-VPL 2019/2020.

In Figure 105 the total number of registrations can be seen (including the user created with the exclusive role of student for the previous test activities).

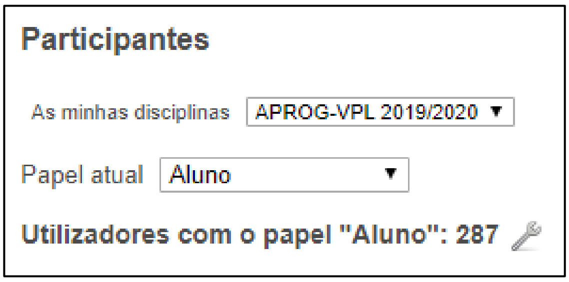


Figure 105 - Total number of students enrolled in APROG-VPL 2019/2020

As mentioned, the exercises have been reorganised and adapted to Mooshak, but being careful to keep them compatible with use in the VPL.

The submission to the VPL continued to be voluntary, with Table 24 showing the number of submissions made in each year, corresponding to an overall average participation rate of 59.8%, considering the universe of 286 students.

Table 24 - VPL submissions in 2019/2020

Exercises	Modularity		Arrays	
	Ex. C	Ex. E	Ex. E	Ex. H
Submissions	183	173	166	162
	64,0%	60,5%	58,0%	56,6%

This school year was not included in the scope of the experiment and no specific questionnaire on the use of VPL was carried out. However, a survey has been set up in the Moodle area of the APROG CU, with some questions whose answers are potentially interesting for the scope of this work, but whose results will not be known in time to be considered here.

5.5 ASSESSMENT OF THE VARIOUS STAGES OF IMPLEMENTATION

The study carried out over the last three academic years was based on the VPL.

Throughout this period, 37 classes were involved in the activities, out of a total of 638 students enrolled and in which 445 used the system, on a voluntary basis, at least once. Eight teachers also participated, while only the author of the study and another teacher were present in all three editions.

In addition to the collaboration of students and teachers, this study required technical interventions at various levels, from administration of Moodle to the configuration of networks and devices.

The actions developed and the activities carried out allowed to deepen the knowledge of the functioning of the VPL, by implementing new verification functionalities in a perspective of improvement and support to the teaching-learning process.

Participants were asked to cooperate by completing online surveys to collect information that could be useful for validating the process.

The following chapter presents an analysis of the main results of this study derived from the results obtained after processing the data collected and after aggregating them to validate the main aspects under study.

6 RESULTS ACHIEVED

"Great achievements are possible when attention is paid to small beginnings".

Lao-Tsé

This chapter presents the results obtained in the course of the study and analyses them. This chapter presents the results obtained in the course of the study together with its analysis.

It begins with a reference to the evaluation of the teaching model used, with a summary description of the study and its objectives.

The mechanisms and processes of data collection used, their characterisation and the reasons for their choice are presented.

The results obtained are presented and analysed, in particular those resulting from the responses to the surveys undertaken.

The chapter concludes with a list of publications produced as part of the study and the presentation of other results and actions arising from this work.

6.1 INTRODUCTION

A work such as the one presented here will have more value if the means of assessing the potential of the model and the proposed prototype are analyzed and studies are carried out on the implementation of the prototype used to support the underlying research.

In this sense, knowing that its evaluation includes the identification of the limitations and explanation of the assumptions for the application of the developed prototypes, it was decided to carry out this validation essentially using two assumptions:

- by direct observation;
- in response to enquiries by those involved in the process.

In addition to these two means, information from the automatic data logging of activities performed in Moodle was also considered.

The data collected from these surveys have been processed statistically with the impartiality required for this type of study, so that appropriate conclusions could be drawn.

6.2 SUMMARY OF OBJECTIVES AND EXPERIMENTAL STUDY

The main objective of this study was to assess the usefulness of using VPL in the teaching-learning process of programming, namely with regard to feedback, enhancing autonomous work. It also aims to verify the potential impact of the use of VPL on reducing the workload of teachers as regards the analysis and evaluation of codification work.

In order to carry out the intended evaluation, an experiment was carried out in the context of DEI-ISEP APROG. To this end, some PL classes were selected according to the availability of collaboration in the experiment by the teachers who taught them. Students in these classes were invited to submit some work on a voluntary basis in the VPL in order to obtain automatic feedback in extra class periods.

6.3 DATA COLLECTION MECHANISMS AND PROCESSES

The evaluation of a process can be carried out on the basis of information obtained from various sources and by using various instruments.

The techniques for collecting information in the field of research may be of the documental or non-documental type, in the latter case they may correspond to direct or indirect observation.

The data obtained in this study came from direct observation during the activities, from automatic records of activities carried out in Moodle and, mainly, from indirect observation through answers to an online questionnaire.

6.3.1 Direct observation

During the study, the teachers supervised and supported the work of the students in carrying out the VPL activities. Although no systematic quantitative records have been made, teachers' perceptions have been considered qualitatively. They were able to see the participation of the students, their enthusiasm and the difficulties encountered, as well as interactions amongst them.

6.3.2 Moodle automatic registers

All actions performed in Moodle were automatically registered and it is possible to obtain a usage profile according to the amount, frequency and time at which accesses were performed.

The information collected in this way was not added together, but was used, fundamentally, to detect any difficulties experienced by each student, allowing timely action to overcome them. It was also considered for the purposes of calculating total access and inferring the interest of using the system.

6.3.3 The use of surveys

As far as indirect observation is concerned, one of the most commonly used options is the survey (Ferreira & Campos, 2009). A survey is the gathering of data about a specific situation, involving several individuals, with the aim to allow generalizations.

The design of a survey requires special care, and the focus of the questions should be on the information to be obtained. It should also be appropriate for potential respondents as regards their age group, literacy, language or other aspects which may be relevant to the specific context. The questions should be objective and should be reviewed by others, preferably with the profile of the recipients, checking that they are clear and easily understandable. A preliminary study or pre-test is also recommended to a limited number of respondents (De Leeuw, Hox, & de Leeuw, 2012).

Current technological means allow surveys to be carried out online (Brečko & Carstens, 2006), bringing various benefits to the process, of which we can highlight (Díaz de Rada & Domínguez-Álvarez, 2014) (Thayer-Hart, Dykema, Elver, Schaeffer, & John, 2010):

- reduction in elaboration time;
- dematerialisation of the process with reduced material and logistical costs;
- potential increase in the number of respondents;
- diversification of response periods;
- increasing the quality and reliability of information;
- centralised data collection and storage;
- easier processing of data;
- possibility of reusability (with or without changes) for new editions.

With online surveys showing a significant number of advantages over the traditional model, there are, however, some less positive aspects, namely (Dorine, Blair, & Preece, 2003):

- potential increase in inconsistent responses;
- potential increase in incomplete responses;
- access dependent on technological means that may not be available.

A very important issue in drawing up the survey, whatever form it takes, is the definition and formulation of the questions, including the scale to be used, in particular as regards the choice of each question.

6.3.4 The scale used: *Likert*

In defining the questions to be included in the survey, it is often necessary to define possibilities for non-dichotomic answers. In such a case, it is usual to use a scale on which to measure the degree of agreement or disagreement with a particular statement.

Although there are several types of scale, the most commonly used scale for concordance analysis is the Likert (Likert, 1932) scale.

This type of scale contains a set of items for each of which the respondent must express their degree of agreement, from full disagreement to full agreement (Figure 106).

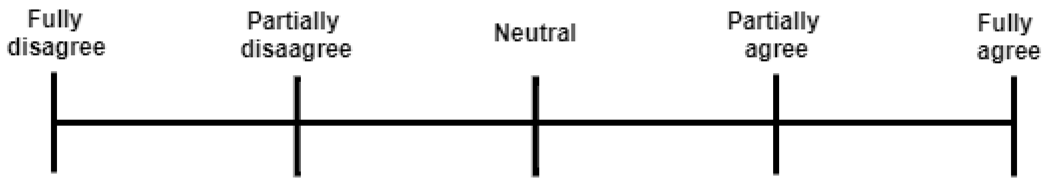


Figure 106 – 5 level *Likert* scale

In the preparation of the surveys, scales of Likert with five levels of choice of questions were preferably used, as originally advocated by its author in 1932.

This scale was chosen taking into account the type of information to be obtained, the number of levels being defined according to the ease and speed of response when compared to a seven-level scale, the degree of reliability being similar (Dalmoro & Vieira, 2013).

To assess the appropriateness of the form and content and the clarity of the questions, pilot tests of the questionnaires were carried out on a limited number of respondents before their final publication, as recommended by Hill (Hill & Hill, 2008).

6.4 Analysis and discussion of results- Student surveys

After the data was collected, it was registered and organised using a spreadsheet. The data collected has been subjected to statistical treatment and analysis, allowing its organization and presentation in a graphical way, allowing a better global view.

As mentioned, the main means of data collection in this study were online (Annex F and Annex G) surveys, which were answered by students. The questions prepared focused mainly on the tool under study as well as questions relating to pedagogical aspects of the experience carried out.

The surveys were purposefully anonymous so as not to condition the responses, boosting presumably more sincere and reliable responses. However, this option carried a risk, identified and accepted beforehand, which consisted in the impossibility of identifying those who had not yet responded, thus making it impossible to reiterate the request for completion targeting those individuals only. Moreover, for the same reason, it would not be possible to cross-check individual respondents' information with their respective academic rank in APROG.

As the main object of the experience was to verify the adequacy of the method and tool used to the teaching-learning process, the option of anonymity was assumed in order to privilege the sincerity of the answers regarding the experience and individual perception of each student, to the detriment of the comparison of results between students who used VPL and students who did not.

Even if such an analysis were possible, the comparison of results would be very much conditioned by other factors that were not and could not be considered, because they were not known or could not be satisfactorily obtained, such as, for example, the individual academic background, possible programming experience or socio-cultural background.

The 2018/2019 survey was improved compared to the previous year, clarifying some questions and adding others that proved to be important, and was answered by a significantly larger number of students.

In this second edition, 217 students and four teachers (from a universe of six) who were directly involved in the pilot experience were enrolled, corresponding to 13 PL classes out of the existing 17.

Of the students involved, 196 actively participated with submissions to the VPL. All students were invited to answer the questionnaire presented in Annex G, and 142 answers were obtained.

6.4.1 Characterization of the participating students

As far as the universe of students is concerned, and as shown in Chart 4, 81% of the respondents were male and the remaining 19% female, which seems to show that computer engineering is in greater demand from male individuals.

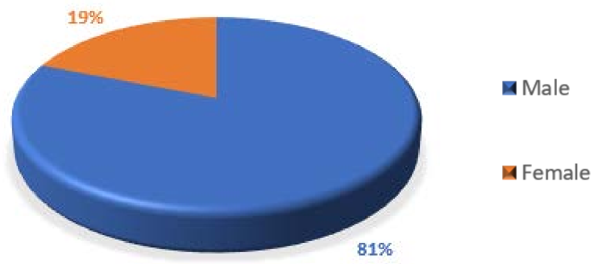


Chart 4 – Distribution of respondents by gender

Regarding the age characterization of the respondents, and as can be seen in Chart 5, 71.8% were aged under 20, 13.4% were aged between 20 and 25 and 14.8% were aged over 25.

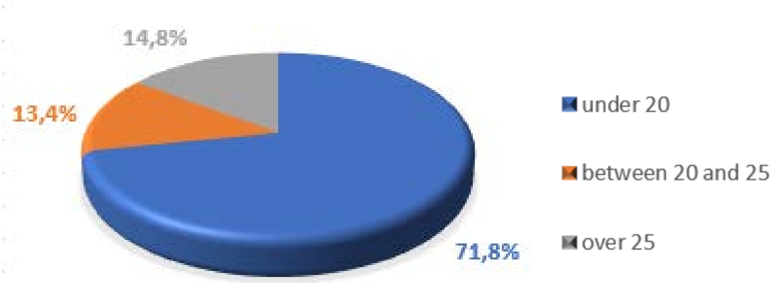


Chart 5 - Age ranges of students

6.4.2 Responses from participants

The survey includes a block with a set of 15 questions related to the profile of the students regarding the use of technology, the preparation of the experience, the use of VPL during the study and the teaching-learning process of programming. In Table 25 we present the questions posed to students.

Table 25 - Questions for students

Question	Description
Q1	Programming is a difficult task
Q2	D-Learning platforms (example Moodle) are an asset
Q3	I would like to have tools to support the resolution of exercises outside the classroom
Q4	The demonstration exercise was useful to become familiar with the VPL
Q5	The use of VPL in the resolution of exercises helped my learning process
Q6	I started using the VPL but lost interest
Q7	The use of VPL is too complicated
Q8	I would have liked to have been able to use the VPL to solve more exercises
Q9	I would like to be able to use the VPL for individual assessment instead of the paper resolution version
Q10	The possibility of resubmission and obtaining automatic grading is very useful
Q11	The pre-set tests were important in identifying shortcomings in my resolutions
Q12	The fact that the number of evaluations was limited had a negative impact on my work
Q13	The VPL is an added value to the teaching-learning process of programming
Q14	The experiment was explained clearly
Q15	Sufficient support has been given for the effective use of VPL

The first three questions were asked from a general perspective, the remaining 12 already focused on VPL and the implementation of the experiment.

The first question was whether respondents considered programming to be a difficult task.

Chart 6 presents the results of the question on the statement "Programming is a difficult task" where it can be seen that 14.1% fully disagrees, 20.4% partially disagrees, 25.4% neither agrees nor disagrees, 25.4% partially agrees and 14.8% fully agrees.

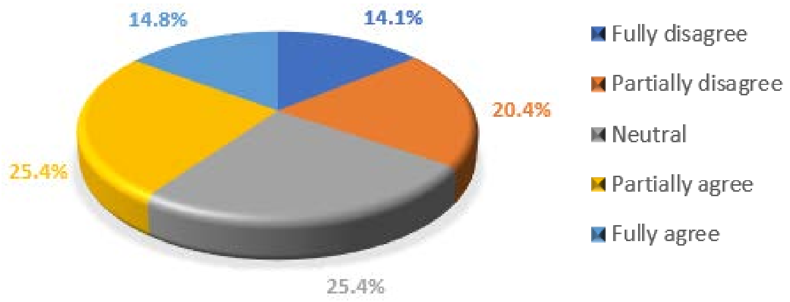


Chart 6 – Q1 - Programming is a difficult task

The analysis of the chart shows that opinions are quite diverse, so it can be inferred that the sample is rather heterogenous, yet, the distribution between those who agree and those who disagree is similar.

Asked about the added value of D-Learning platforms, Chart 7 shows that the majority of students (62.7%) fully agree with the statement. It was also possible to observe that 27.5% said they partially agreed, 5.6% neither agreed nor disagreed and 4.2% partially disagreed.

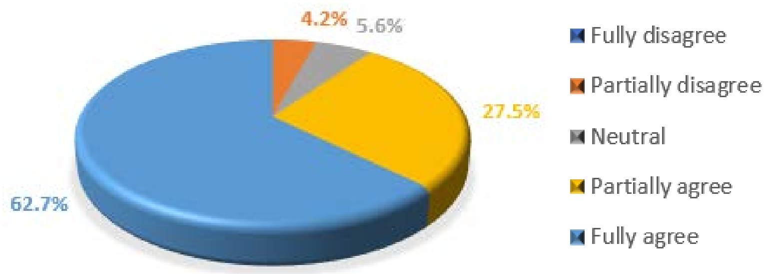


Chart 7 – Q2 - Added value of using distance learning platforms

The analysis of the graph allows us to assess that the opinions are frankly favourable to the use of an LMS to support the teaching-learning process.

After the experiment and questioned whether "they would like to have tools to support the resolution of exercises outside the classroom", the distribution of the answers is as presented in Chart 8.

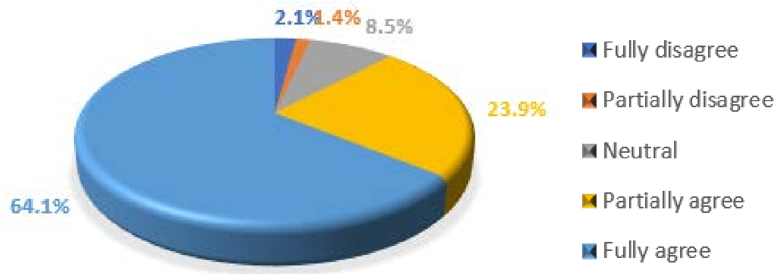


Chart 8 – Q3 - I would like to have tools to support the resolution of exercises outside the classroom

As can be observed, the majority of students (64.1%) say yes and 23.9% partially agree, which makes a total of 88.0% of respondents pointing out that they would like to have tools to support the resolution of exercises outside the classroom.

Before the process began, a demonstration exercise was carried out to familiarise the students involved with the use of the VLP. Asked about its usefulness they answered as follows (Chart 9).

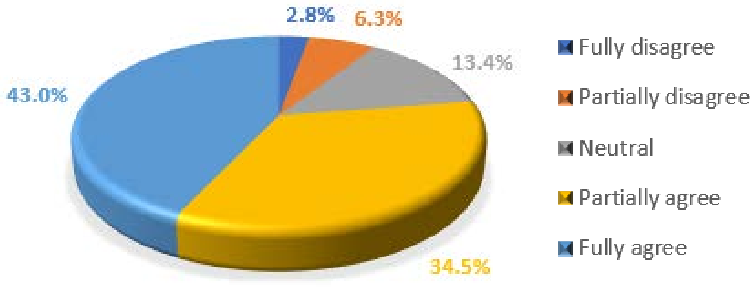


Chart 9 - Q4 - The demonstration exercise was useful to familiarize you with the VPL

The results show that 77.5% of the respondents considered this activity useful in carrying out the experiment.

Students were asked about the usefulness of VPL in the teaching-learning process of programming to try to assess its potential.

Most students (56.3%) agree (fully or partially) with the statement. Of those surveyed, 22.5% still did not have an opinion, so they neither agreed nor disagreed and 21.1% said they disagreed partially (14.1%) or totally (7.0%) (Chart 10) with the statement.

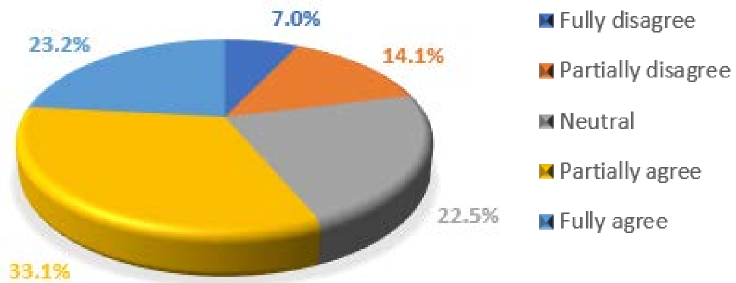


Chart 10 - Q5 - Use of VPL in the teaching-learning process of programming

The analysis of the chart shows that the use of VPL is positive although it should be made more motivating and appealing.

The possible decrease in students' interest in VPL throughout the process was also analysed. The results indicate that most students maintained interest in the VPL throughout the study (Chart 11).

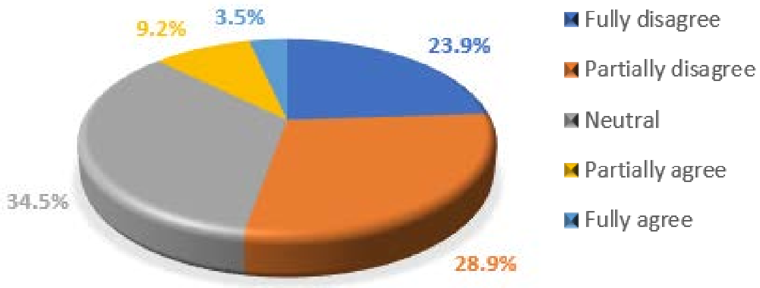


Chart 11 - Q6 - I started using the VPL but lost interest

Although only 12.7% of students were demotivated, this is something that needs to be researched to reduce this value in the future.

A question has been drawn up to assess the simplicity or complexity of students' use of VPL.

As can be seen in Chart 12, 24.6% of respondents agree (fully or partially).

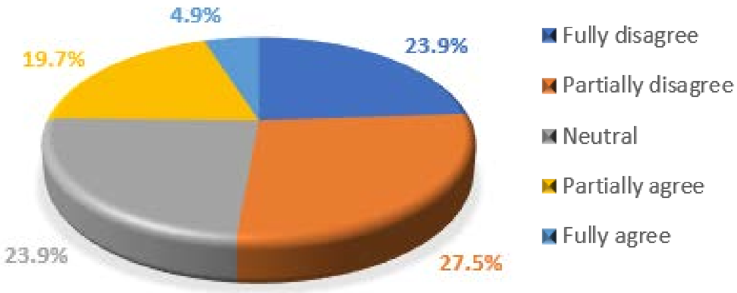


Chart 12 - Q7 - The use of VPL is too complicated

Although 75.4% did not experience difficulties in using the VPL, the process can be improved to make it even simpler.

As some students were asked to use the VPL to submit more exercises than those initially available, a question was included in the questionnaire to obtain the general opinion of the students about that possibility.

In Chart 13 it may be seen that 23.2% of the responses are discordant, with 44.4% of students stating this intention.

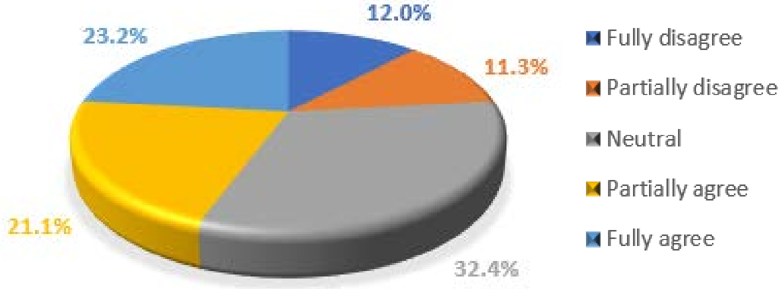


Chart 13 - Q8 - I would have liked to have been able to use the VPL to solve more exercises

The results obtained, which are not conclusive in terms of a general willingness to increase the use of VPL, indicate that most students agree or have no opinion on this reality.

Individual paper resolution being the current practice for the assessment of codification exercises, and with the VPL having adequate resources for assessment contexts, students were asked about the use of the VPL for this purpose.

The results presented in Chart 14 show that 66.9% of students manifest this preference, while 18.3% remain more interested in the traditional method, i.e. paper-based evaluation.

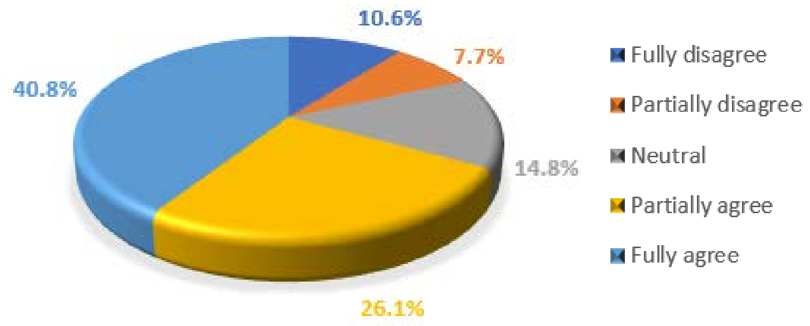


Chart 14 – Q9 - I would like to be able to use the VPL for individual assessment instead of the paper-based version

These results are in line with the opinions expressed verbally by students at various times during the semester.

In the running of the VPL it is possible to obtain a rating and, if it does not correspond to the maximum value, the system presents the results of the failed tests, prompting the student to understand what needs to be corrected.

As automatic grading is one of the VPL's functionalities, we wanted to know the students' opinion about its usefulness, associated to the possibility of resubmission of works (Chart 15).

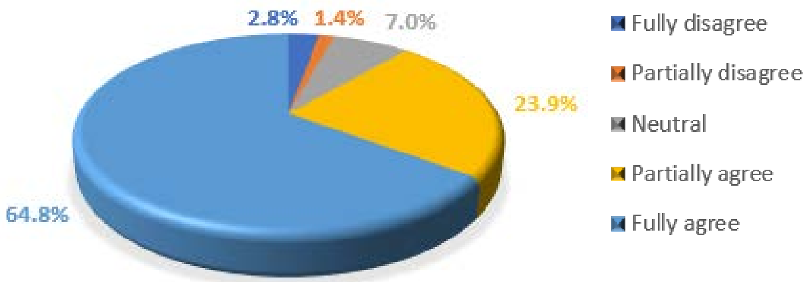


Chart 15 – Q10 - The possibility of resubmission and obtaining automatic classification is very useful

From the results obtained, the usefulness of these functionalities is clear, with 88.7% of students expressing this opinion.

A question was asked about the importance students attributed to pre-defined test cases for the detection of coding deficiencies.

Chart 16 shows that most students consider the tests to be important for the above objective, with 45.1% fully agreeing and 33.8% partially agreeing with the statement.

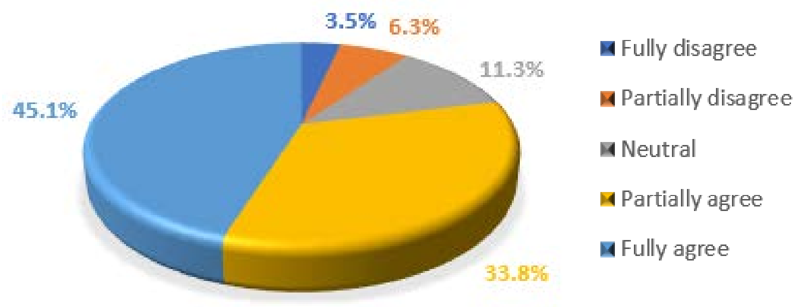


Chart 16 - Q11 - The pre-defined tests were important in identifying weaknesses in my resolutions

Because the test cases are designed to verify the correct functioning of the programmes, allowing the detection of logical or conceptual errors, students' responses are in line with initial expectations.

As mentioned above and the reasons explained, the number of submissions was deliberately limited in some activities, so it was intended to ascertain whether this aspect had been considered negative by students.

The responses were fairly evenly spread among the various options, with only 7.7% of respondents fully agreeing with the statement (Chart 17).

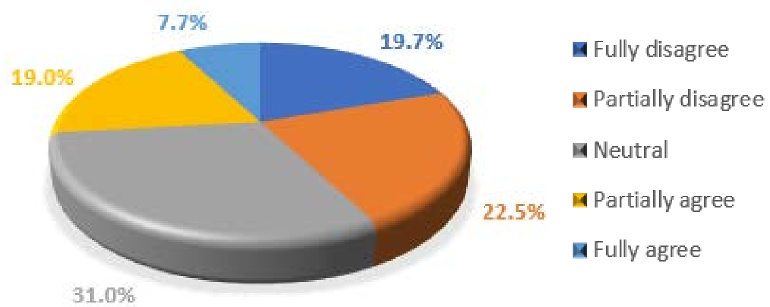


Chart 17 - Q12 - The fact that the number of assessments was limited had a negative impact on my work

The fact that 26.8% of respondents considered the limitation of submissions to be a negative aspect may be due to the change introduced at the end of the study. This change was the result of the discovery of the use of a strategy of trial-error in the way of solving exercises.

The intention was also to gather the opinion on the added value of VPL for the teaching-learning process of programming.

Only 9.1% of students responded negatively to this question (Chart 18).

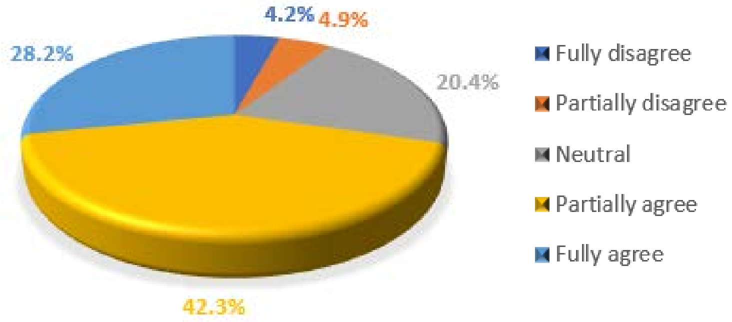


Chart 18 - Q13 - The VPL is an added value to the teaching-learning process of programming

The high rate of positive responses (70.4%) is noteworthy. This figure is all the more relevant given that only 34.5% of respondents did not agree with the statement in response to question Q1 (Programming is a difficult task).

As mentioned above, a prior explanation of the experiment to be carried out, its objectives and the way in which it was achieved was given. The aim was to find out the students' opinion on the clarity of the explanations made.

In Chart 19 it is possible to observe that 70.4% of the respondents agree fully or partially with the clarity of the explanation.

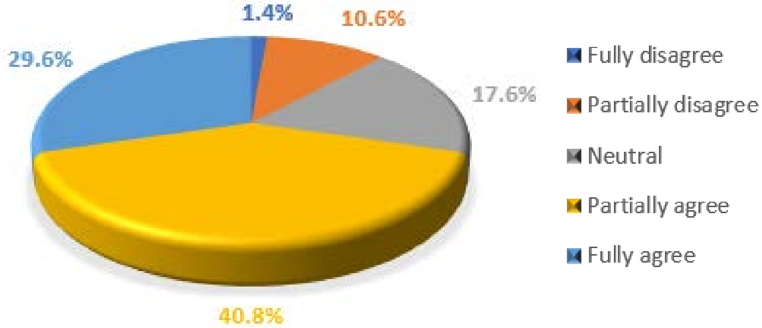


Chart 19 - Q14 - The experiment was clearly explained

The results mostly pointed to the clarity of the explanation, with only 1.4% (2 students) fully disagreeing.

As for the conditions of use of the VPL, students were asked whether the support given for this purpose had been enough. 67.6% of the respondents answered positively (40.1% agreeing fully and 27.5% partially)Chart 20).

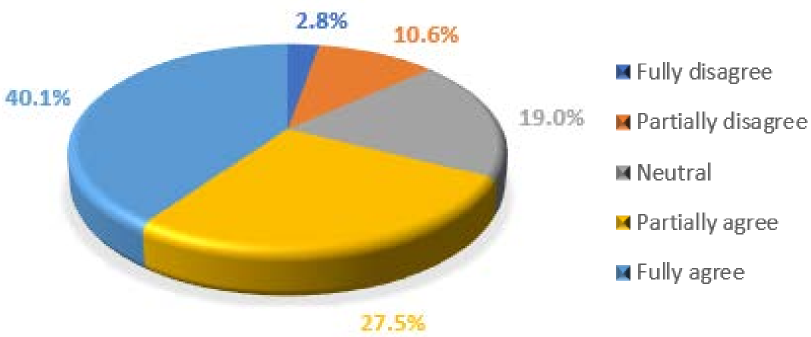


Chart 20 - Q15 - Sufficient support has been given for the effective use of VPL

The support given is considered to have been satisfactory overall. However, the process could be improved and should be an aspect for future consideration.

In Table 26 the overall results of the answers to the questions mentioned are presented globally.

Table 26 - Global results

Questions	Fully disagree	Partially disagree	Neutral	Partially agree	Fully agree
Q1	14,1%	20,4%	25,4%	25,4%	14,8%
Q2	0,0%	4,2%	5,6%	27,5%	62,7%
Q3	2,1%	1,4%	8,5%	23,9%	64,1%
Q4	2,8%	6,3%	13,4%	34,5%	43,0%
Q5	7,0%	14,1%	22,5%	33,1%	23,2%
Q6	23,9%	28,9%	34,5%	9,2%	3,5%
Q7	23,9%	27,5%	23,9%	19,7%	4,9%
Q8	12,0%	11,3%	32,4%	21,1%	23,2%
Q9	10,6%	7,7%	14,8%	26,1%	40,8%
Q10	2,8%	1,4%	7,0%	23,9%	64,8%
Q11	3,5%	6,3%	11,3%	33,8%	45,1%
Q12	19,7%	22,5%	31,0%	19,0%	7,7%
Q13	4,2%	4,9%	20,4%	42,3%	28,2%
Q14	1,4%	10,6%	17,6%	40,8%	29,6%
Q15	2,8%	10,6%	19,0%	27,5%	40,1%

6.4.3 Analysis of the relevance of the results

To perform the desired analysis, the answers need to be numerical, so a Likert scale of five levels has been used and each of the options has been assigned numerical values (Table 27).

Table 27 - Numerical values of Likert's five-level scale

Option	Numeric value
Fully disagree	1
Partially disagree	2
Neutral	3
Partially agree	4
Fully agree	5

In order to assess the level of agreement of the questions in comparative terms, a score of agreement was determined for each item, in percent, calculated by the quotient of the total points attributed to the respective item and the maximum possible value.

$$Score = \frac{Total\ item\ points}{Maximum\ possible\ value} \times 100\%$$

In Chart 21 the scores calculated for each of the questions are shown in descending order according to students' answers.

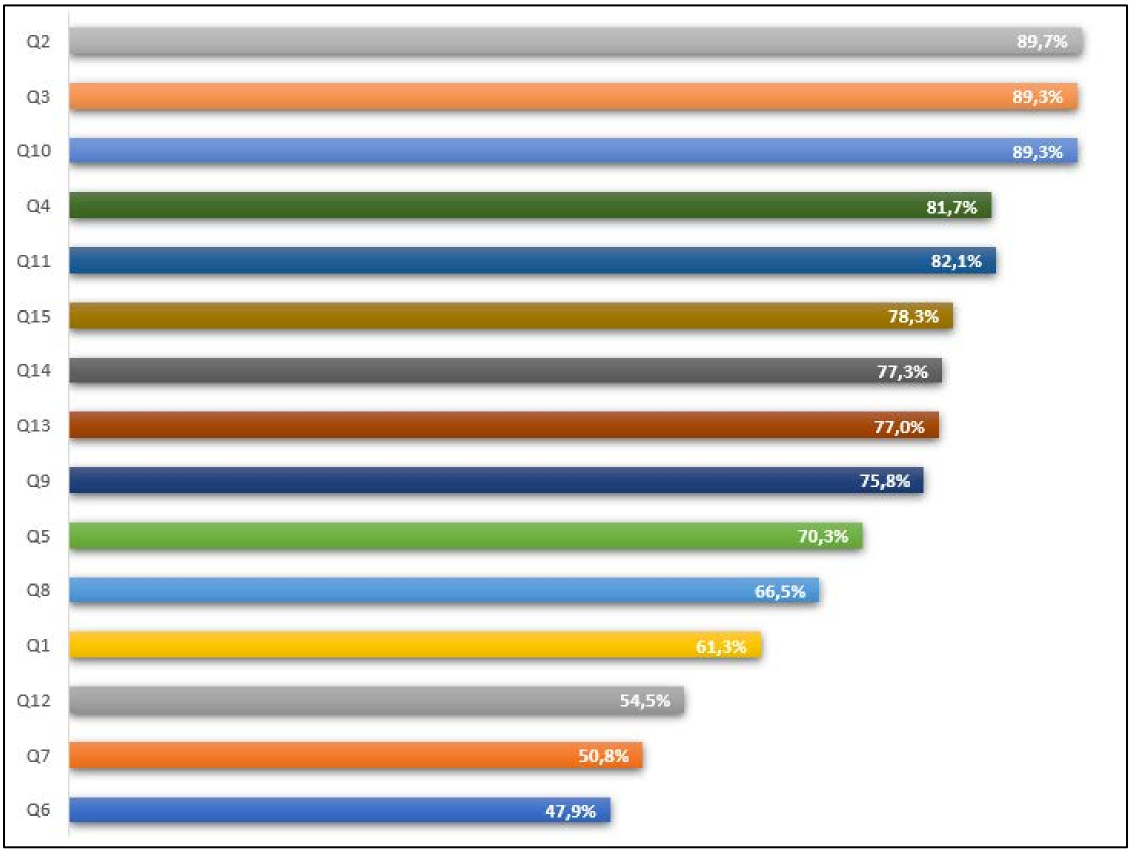


Chart 21 - Score of agreement on the questions raised

Questions Q2 (D-Learning platforms (example Moodle) are an asset), Q3 (I would like to have tools to support the resolution of exercises outside the classroom) and Q10 (The possibility of resubmissions and obtaining an automatic grade is very useful) are those to which students agree with the most.

These results indicate willingness to use the technologies, as well as interest in the use of tools that enhance automation and autonomous work.

On the other end are the results for questions Q6 (I started using the VPL but lost interest), Q7 (The use of VPL is too complicated) and Q12 (The fact that the number of evaluations was limited had a negative impact on my work), with the lowest score of agreement. These issues are related to less positive assessments so that the lowest score contributes to an overall favourable assessment of the system.

Using the same conversion from the original scale to numerical values, the mean, median, standard deviation and coefficient of variation of each question were calculated (Table 28).

Table 28 – Statistical response figures

Question	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
Mean	3,06	4,49	4,46	4,08	3,51	2,39	2,54	3,32	3,79	4,46	4,11	2,73	3,85	3,87	3,92
Median	3,00	5,00	5,00	4,00	4,00	2,00	2,00	3,00	4,00	5,00	4,00	3,00	4,00	4,00	4,00
Standard deviation	1,27	0,79	0,87	1,03	1,20	1,06	1,19	1,28	1,34	0,90	1,06	1,20	1,02	1,01	1,13
Coefficient of variation	42%	18%	20%	35%	34%	44%	47%	39%	35%	20%	26%	44%	27%	26%	29%

As can be seen, the highest standard deviations correspond to questions Q9, Q8, Q1, Q12 and Q5 highlighting the dispersion of opinions in these answers.

The answers to questions Q1, Q6, Q7 and Q12 show the highest values of the coefficient of variation corresponding to the greatest heterogeneity of answers, whereas questions Q2, Q3 and Q10 show the greatest homogeneity.

The answers to questions Q2 and Q3 are those with the highest concordance as can be seen from the maximum median value.

The lowest median values correspond to questions Q6 and Q7, indicating low levels of agreement, which demonstrates students' interest in VPL.

From the overall analysis of the results it is possible to conclude that the initial characterization of the profile of respondents shows a marked willingness to use technologies, associated with the possibility of enabling means of autonomous study. This was an expected result, as they are students in a computer engineering course. More surprising are the results regarding the perception of difficulty of programming activity (Q1) where there is a great diversity of opinions.

As far as the use of VPL is concerned, the students did not reveal any significant difficulties, but still showed considerable interest and enthusiasm in the course of the experiment, as can be inferred from the results, particularly in the answers to questions Q6 (I started using the VPL but lost interest), Q7 (The use of VPL is too complicated) and Q10 (The possibility of resubmission and obtaining automatic classification is very useful).

Regarding the teaching-learning process the overall appreciation by the students is very positive and there is a low degree of disagreement on the statements of Q13 (VPL is an added value to the teaching-learning process of programming) and Q5 (The use of VPL in the resolution of exercises was an aid to my learning process). These are very rewarding results for the effort and commitment dedicated to the study and motivators for the future use of VPL in teaching programming.

The results concerning the preparation of the experiment, in particular questions Q4 (The demonstration exercise was useful in familiarising students with the VPL), Q11 (The pre-defined tests were important in identifying deficiencies in my resolutions), Q14 (The experiment was explained clearly) and Q15 (Enough support was given for the effective use of the VPL), indicate that it was adequate, providing students with the necessary conditions for the development of activities.

6.4.4 Evaluation of the effect of socio-demographic variables

To assess the possible impact of gender, age and some other factors on the variations in responses, we performed the Mann-Whitney (non-parametric) test for independent samples. If the p-value (significance) obtained from the sample distribution on the null hypothesis is lower than 0.05, then there are significant differences in responses according to the respondent group.

In terms of gender, the test showed significant differences in the answers to question Q1 (Programming is a difficult task) (Table 29).

Table 29 – Mann-Whitney gender test results

Gender		
Question	Mann-Whitney U	p-value
Q1	971,500	0,002

In Table 30 we present the values of the mean and median for question Q1 in each of the groups considered.

Table 30 – Mean, median and standard deviation according to gender

Gender	Female			Male		
n	38			104		
Question	Mean	Median	Standard deviation	Mean	Median	Standard deviation
Q1	3,74	4,00	1,16	2,90	3,00	1,25

The impact of gender on the response to question Q1 (Programming is a difficult task) is significant, with female respondents tending to consider programming more difficult than male respondents. This impact is evident in the differences in response values, with a median of 4.00 and a mean of 3.74 for female, as opposed to 3.00 and 2.90 for male.

The aim was also to see how the age of respondents could lead to different responses. In the survey there were three response options for the age group (under 20, between 20 and 25 and over 25), with 71.8% of respondents being under 20. For this reason, in order to make the sample group size more balanced, the other two age groups were grouped together for the purpose of applying the Mann-Whitney test, and the results presented in Table 31 were obtained.

Table 31 – Mann-Whitney age group test results

Age range		
Question	Mann-Whitney U	p-value
Q1	1552,500	0,024
Q10	1652,000	0,037

The values of descriptive measures were calculated, which are shown in Table 32.

Table 32 – Age-related descriptive measures

Age	< 20 years old			>= 20 years old		
n	102			40		
Question	Mean	Median	Standard deviation	Mean	Median	Standard deviation
Q1	3,22	3,00	1,26	2,68	3,00	1,23
Q10	4,54	5,00	0,86	4,28	4,50	0,99

The age factor impacts the answers to questions Q1 (programming is a difficult task) and Q10 (the possibility of resubmission and obtaining automatic classification is very useful). It can be observed that younger students tend to find the task of programming more difficult than their older colleagues, with an average of 3.22, compared to 2.68 in response to question Q1. With regard to the automatic grading, younger students find it more useful than their older colleagues, the mean values being 4.54 and 4.28 and the median values 5.00 and 4.50 respectively, and there are no marked differences in the standard deviations.

The results are further clustered by considering two distinct groups according to the response to the existence of prior algorithmic knowledge before starting to attend APROG. To the question "Before you started APROG, did you have any knowledge of algorithmics?" 61.3% of respondents answered positively.

The results of these groups' analysis, applying the Mann-Whitney test, indicate significant differences in the answers to questions Q1 (Programming is a difficult task) and Q9 (I would like to be able to use the VPL for individual assessment instead of the paper based version), as shown in Table 33.

Table 33 – Mann-Whitney test results for prior knowledge of algorithms

Previous knowledge of algorithmics		
Question	Mann-Whitney U	p-value
Q1	1750,500	0,006
Q9	1921,000	0,038

Table 34 – Descriptive measures based on prior knowledge of algorithmics

Previous knowledge of algorithmics	yes			no		
n	87			55		
Question	Mean	Median	Standard deviation	Mean	Median	Standard deviation
Q1	2,83	3,00	1,27	3,44	4,00	1,20
Q9	3,97	4,00	1,28	3,51	4,00	1,39

As can be seen in Table 34 previous knowledge of algorithmics strongly influenced the responses regarding the perception of the difficulties of the task of programming, and the perception of greater difficulty of those without previous experience was evident. In the group of those who answered the question affirmatively, the mean and median values of the answers to question Q1 (Programming is a difficult task) were 2.83 and 3.00 respectively. For the remaining ones, the mean and median values of the answers to question Q1 were 3.44 and 4.00 respectively, the standard deviations being roughly of the same order of magnitude.

With regard to question Q9 (I would like to be able to use the VPL for individual evaluation in place of the paper-based version), in the group that had previous knowledge of algorithms, the mean was 3.97 and 3.51 in the other group, while the median was the same in both groups with a value of 4.00.

The answers for those who responded differently to the question "Before you started attending APROG, did you have any programming experience?" were also analysed. In this case the groups were very similar in scale with 51.4% positive and 48.6% negative responses.

The Mann-Whitney test applied to these groups revealed significant differences in answers to questions Q1 (Programming is a difficult task) and Q13 (VPL is an added value to the programming teaching-learning process), although in this case, on the threshold of negating the null hypothesis (p-value = 0.054) as can be seen in Table 35.

Table 35 – Results of the Mann-Whitney test on prior programming knowledge

Previous knowledge of programming		
Question	Mann-Whitney U	p-value
Q1	1783,500	0,002
Q13	2073,00	0,054

Table 36 – Descriptive measures based on prior knowledge of programming

Previous knowledge of algorithmics.	yes			no		
n	73			69		
Question	Mean	Median	Standard deviation	Mean	Median	Standard deviation
Q1	2,75	3,00	1,19	3,39	4,00	1,29
Q13	4,00	4,00	0,99	3,70	4,00	1,05

The answers to question Q1 (Programming is a difficult task) from the group that already had previous knowledge of programming resulted in values of 3.39 and 4.00 for the mean and median, respectively, the other group being 2.75 and 3.00 respectively (Table 36). It is clear that previous programming experience promotes a less difficult perception of the task of programming. This can be an important conclusion and can help to de-mystify the complexity associated with the task of programming by enhancing better learning.

Regarding question Q13 (the VPL is an added value to the programming teaching-learning process) there is an impact of the programming experience effect. Those with previous experience value this question more, with an average of 4.00 answers and 3.70 in the other group.

It was also intended to identify possible differences in perception between the groups of those who carried out all the proposed exercises and the others. Thus, those who submitted less than six exercises were grouped, corresponding to 26.8% of the respondents, with 73.2% carrying out all the tasks.

The application of the Mann-Whitney test to these groups identified significant differences in the answers to questions Q4 (The demonstration exercise was useful in familiarising me with VPL), Q12 (The limited number of assessments had a negative impact on my work) and Q15 (Sufficient support was given for the effective use of VPL Table).

Table 37 – Mann-Whitney test results for the number of exercises submitted

Number of exercises submitted		
Question	Mann-Whitney U	p-value
Q4	1750,500	0,006
Q12	1750,500	0,006
Q15	1921,000	0,038

In Table 38 the values of the means, medians and standard deviations for each of the questions mentioned and in each of the groups considered are presented.

Table 38 – Descriptive measures depending on the number of exercises submitted

Number of exercises	<6			6		
n	38			104		
Question	Mean	Median	Standard deviation	Mean	Median	Standard deviation
Q4	3,74	4,00	0,98	4,21	4,50	1,03
Q12	3,08	3,00	1,22	2,60	3,00	1,18
Q15	3,63	4,00	1,13	4,02	4,00	1,11

As can be observed, the answers to questions Q4 and Q15 present higher means in the group of students who submitted all the exercises. The fact that they were more actively involved may indicate that they would be more committed to the process and probably have better informed opinions. Regarding the answers to question Q12 (the fact that the number of evaluations was limited had a negative impact on my work), the mean value of the group that submitted all the exercises indicates a fairly neutral opinion (2.60), while the mean value of the other group was 3.08, pointing to an overall opinion more in agreement with the statement.

6.4.5 Evaluation of the effect of perceiving the difficulty of the task of programming

Since question Q1 (Programming is a difficult task) presents significant differences between the different groups in the analyses made and previously presented, it was decided to carry out an analysis of question Q1 in order to understand the effect of the answers to this question in the answers to the others.

As the Likert scale is used at five levels, the answers were grouped into three groups to simplify the analysis. For this purpose, one group (A) was considered to contain the elements that fully or partially disagree with the statement, a second group (B) for those who expressed an opinion of neutrality and, finally, group (C) for those who consider programming a difficult task (Table 39).

Table 39 – Grouping of answers to question Q1

Group	*Likert* scale levels	Respondents	
A	Fully disagree Partially disagree	49	34,5%
B	Neutral	36	25,4%
C	Partially agree Fully agree	57	40,1%

As the Mann-Whitney test applies to only two independent samples, and there are three groups in this case, the non-parametric Kruskal-Wallis test was applied, which is an extension of the Mann-Whitney test (Maroco & Garcia-Marques, 2006).

Comparing groups A and B it can be seen that the only issue where statistically significant differences exist is in question Q6 (I started using VPL but lost interest) (Table 40).

Table 40 – Results of the Kruskal-Wallis test on question Q1 (Groups A and B)

Programming is a difficult task		
Question	**Kruskal-Wallis H**	**p-value**
Q6	650,000	0,037

Table 41 – Descriptive measures relating to Question Q1 (Groups A and B)

Group	A			B		
Question	**Mean**	**Median**	**Standard deviation**	**Mean**	**Median**	**Standard deviation**
Q6	2,37	2,00	**1,11**	1,86	2,00	**0,90**

Although the median is the same in both groups (2.0), the test result indicates significant differences, explained by means of 2.37 and 1.86 in groups A and B, and also by relatively different standard deviations. These figures indicate that students who initially had less difficulty in the task of programming felt less motivated to use the VPL throughout the process.

The comparison between groups A and C shows the statistically significant differences presented in Table 42.

Table 42 – Results of the Kruskal-Wallis test on question Q1 (Groups A and C)

Programming is a difficult task		
Question	**Kruskal-Wallis H**	**p-value**
Q6	1094,500	0,045
Q10	1102,500	0,026
Q11	1066,500	0,024
Q12	996,500	0,009

Table 43 shows the figures for the descriptive measures of the answers to the questions with statistically significant differences, by groups A and C.

Table 43 – Descriptive measures concerning question Q1 (Groups A and C)

Group	A			C		
Question	**Mean**	**Median**	**Standard deviation**	**Mean**	**Median**	**Standard deviation**
Q6	2,37	2,00	1,11	2,75	3,00	0,97
Q10	4,22	5,00	1,12	4,61	5,00	0,82
Q11	3,84	4,00	1,30	4,37	5,00	0,88
Q12	2,49	3,00	1,23	3,12	3,00	1,23

The answers to question Q6 (I started using VPL but lost interest) by group C (students with more difficulty perceiving the programming task) indicate that the members of this group lost interest in the process of using VPL more easily than their colleagues in group A. This conclusion is in line with the feedback obtained by direct observation during the experiment.

Regarding question Q10 (The possibility of resubmission and obtaining automatic classification is very useful), there is a proximity of opinions between the two groups, the median of both being equal and with the value 5.0. This figure shows that both groups perceive the functionality under evaluation as very positive, with the responses from group C showing a higher mean value than group A.

Q11 (The pre-defined tests were important in identifying deficiencies in my resolutions), showed greater agreement in group C, with a mean of 4.37 and a median of 5.0, while the values in group A were 3.84 for the mean and 4.0 for the median.

Also on question Q12 (the fact that the number of evaluations was limited had a negative impact on my work), although the median was the same in both groups, the mean value was higher in group C, being 3.12, while in group A it was 2.49.

From an overall analysis of these issues concerning the comparison of groups A and C, it is clear that, in general terms, the students, who perceive the task of programming as being difficult, value the use of VPL more. This is an important conclusion because, presumably the most difficult to develop, they may have an ally in the VPL to help them develop their codification skills.

In the Kruskal-Wallis test there are statistically significant differences between the answers of groups B and C in questions Q3, Q6 and Q12 (Table 44).

Table 44 – Results of the Kruskal-Wallis test on question Q1 (Groups B and C)

Programming is a difficult task		
Question	Kruskal-Wallis H	p-value
Q3	732,000	0,006
Q6	536,000	0,000
Q12	672,500	0,004

Table 45 presents the descriptive measures of the answers to question Q1 by groups B and C.

Table 45 – Descriptive measures relating to Question Q1 (Groups B and C)

Group	B			C		
Question	Mean	Median	Standard deviation	Mean	Median	Standard deviation
Q3	4,28	4,50	0,81	4,72	5,00	0,49
Q6	1,86	2,00	0,90	2,75	3,00	0,97
Q12	2,42	2,00	0,97	3,12	3,00	1,23

On question Q3 (I would like to have tools to support the resolution of exercises outside the classroom) there is a significant degree of agreement with the statement by both groups, with evidence for group C where both the mean and the median show higher values.

The answers to question Q6 (I started by using VPL but lost interest) show, out of curiosity, that students who are neutral about the difficulty of programming are less likely to be disinterested than everyone else.

Regarding question Q12 (the fact that the number of assessments was limited had a negative impact on my work), it can be seen that the respondents in group C (students with more difficulty perceiving the programming task) see this as more limiting than their colleagues in group B, with significant differences in all the values considered.

6.4.6 Nonparametric correlations

As mentioned above, the survey conducted in 2018/2019 was substantially improved compared to the previous year.

The answers were obtained after the conclusion of the APROG classes and up to the final examination, the survey being available between 2018/12/24 and 2019/01/17.

Since questions Q6, Q7 and Q12 were formulated in reverse order to analyze the internal consistency of the questionnaire, the answers to these questions were inverted. Question Q7 was the same in both years, with Question Q6 from 2018/2019 corresponding to Question Q5 from the previous year. Thus, in 2017/2018 the results of Q5 and Q7 were reversed.

After this reversal, the survey data was analyzed using Microsoft Excel and IBM SPSS Statistics (version 25).

In order to investigate possible associations between the questions we determined Spearman's non-parametric correlations matrix based on the numerical values associated to each of the options presented in Table 27.

The higher or lower strength of a linear relationship between two variables is measured by a correlation coefficient (Dancey & Reidy, 2011), which can vary between 0 and 1 or between -1 and 0, if the correlation is positive or negative, respectively. Higher values (in absolute value) correspond to stronger ratios, as shown in Table 46 for the rho coefficients (ρ) of Spearman.

Table 46 – Strength of correlation as a function of Spearman's coefficient

(Dancey & Reidy, 2011)

Spearman (ρ)	Correlation
$\rho \geq 0,7$	very strong
$0,4 \leq \rho < 0,7$	strong
$0,3 \leq \rho < 0,4$	moderate
$0,2 \leq \rho < 0,3$	weak
$\alpha < 0,2$	negligible

In addition to the coefficient value, the significance (p-value) must be taken into account, which must be under 0.05 in order to reject the null hypothesis of independence, which is the basis of the algorithm implemented in the SPSS for conducting the test.

Table 47 - Spearman's Matrix of Non-Parametric Correlations

		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
Q2	ρ	0,032													
	sig.	(0,707)													
Q3	ρ	0,159	0,227**												
	sig.	(0,059)	(0,007)												
Q4	ρ	0,031	0,184*	0,163											
	sig.	(0,715)	(0,029)	(0,053)											
Q5	ρ	0,092	0,295**	0,246**	0,429**										
	sig.	(0,277)	(0,000)	(0,003)	(0,000)										
Q6	ρ	-0,170	0,027	0,030	0,290**	0,379**									
	sig.	(0,043)	(0,746)	(0,722)	(0,000)	(0,000)									
Q7	ρ	-0,064	0,174*	0,039	0,353**	0,378**	0,260**								
	sig.	(0,447)	(0,038)	(0,649)	(0,000)	(0,000)	(0,002)								
Q8	ρ	-0,013	0,126	0,129	0,245**	0,511**	0,289**	0,391**							
	sig.	(0,879)	(0,136)	(0,127)	0,003	(0,000)	(0,000)	(0,000)							
Q9	ρ	0,186*	0,192*	0,230**	0,212*	0,244**	0,174*	0,165	0,313**						
	sig.	(0,027)	(0,022)	(0,006)	(0,011)	(0,003)	(0,039)	(0,05)	(0,000)						
Q10	ρ	0,207*	0,222**	0,199*	0,408**	0,365**	0,265**	0,253**	0,299**	0,539**					
	sig.	(0,013)	(0,008)	(0,018)	(0,000)	(0,000)	(0,001)	(0,002)	(0,000)	(0,000)					
Q11	ρ	0,223**	0,283**	0,311**	0,413**	0,497**	0,255**	0,289**	0,365**	0,376**	0,491**				
	sig.	(0,008)	(0,001)	(0,000)	(0,000)	(0,000)	(0,002)	(0,000)	(0,000)	(0,000)	(0,000)				
Q12	ρ	-0,223**	0,168*	-0,170*	-0,032	-0,100	0,223**	0,164	0,015	-0,025	0,034	-0,013			
	sig.	(0,008)	(0,045)	(0,043)	(0,704)	(0,234)	(0,008)	(0,051)	(0,86)	(0,766)	(0,692)	(0,875)			
Q13	ρ	-0,001	0,316**	0,223**	0,442**	0,689**	0,319**	0,436**	0,607**	0,331**	0,418**	0,462**	-0,020		
	sig.	(0,987)	(0,000)	(0,008)	(0,000)	(0,000)	(0,000)	(0,000)	(0,000)	(0,000)	(0,000)	(0,000)	(0,814)		
Q14	ρ	-0,116	0,260**	0,083	0,490**	0,342**	0,202*	0,374**	0,327**	0,193*	0,244**	0,404**	-0,028	0,505**	
	sig.	(0,169)	(0,002)	(0,324)	(0,000)	(0,000)	(0,016)	(0,000)	(0,000)	(0,021)	(0,003)	(0,000)	(0,743)	(0,000)	
Q15	ρ	-0,046	0,228**	0,060	0,465**	0,266**	0,193*	0,407**	0,236**	0,136	0,288**	0,301**	0,105	0,395**	0,686**
	sig.	(0,585)	(0,006)	(0,475)	(0,000)	(0,001)	(0,021)	(0,000)	(0,005)	(0,106)	(0,001)	(0,000)	(0,214)	(0,000)	(0,000)

* Correlation is significant at the 0.05 level (2-tailed).

** Correlation is significant at the 0.01 level (2-tailed).

Table 47 shows the values of the Spearman correlation coefficients (ρ) of the 2018/2019 survey questions, as well as the corresponding significance, with meaningful correlations. From Spearman's matrix of non-parametric correlations some significant associations between some pairs of questions can be observed (Table 48).

Table 48 – Significant peer-to-peer associations of issues

		Spearman Coefficient (ρ)	p-value
Q4	Q5	0,429**	0,000
	Q10	0,408**	0,000
	Q11	0,413**	0,000
	Q13	0,442**	0,000
	Q14	0,490**	0,000
	Q15	0,465**	0,000
Q5	Q8	0,511**	0,000
	Q11	0,497**	0,000
	Q13	0,689**	0,000
Q7	Q13	0,436**	0,000
	Q15	0,407**	0,000
Q8	Q13	0,607**	0,000
Q9	Q10	0,539**	0,000
Q10	Q11	0,491**	0,000
	Q13	0,418**	0,000
Q11	Q13	0,462**	0,000
	Q14	0,404**	0,000
Q13	Q14	0,505**	0,000
Q14	Q15	0,686**	0,000

From the analysis of Table 48 it appears that the moderately significant correlations of the Q4 question (The demonstration exercise was useful for the familiarisation with the VPL) with the others presented are explained to the extent that these questions are related to the use of the VPL and, in particular, to the course of the experiment.

By way of example, the value of 0.490 of the correlation Q4/Q14 (The experience was explained clearly) can be mentioned, and it is clear that there is a direct relationship between the clarity of the explanation of the experience and the usefulness of the demonstration exercise for this purpose.

Referring to the strongest correlations found, a rho of 0.686 in the Q14/Q15 correlation and 0.689 in the Q5/Q13 correlation, respectively, can be observed.

This strong correlation between Q5 (The use of VPL in the resolution of exercises was helpful for my learning process) and Q13 (VPL is an asset for the programming teaching-learning process) shows that those who value the use of VPL for the programming teaching-learning process also understand that its use was useful for learning which seems coherent.

The correlation between questions Q14 (The experiment was explained clearly) and Q15 (Sufficient support was given for the effective use of VPL) indicates coherence since there is agreement between the clarity of the explanation and the support given for the implementation, and since both questions concern the action of the teachers.

Question Q8 (I would have liked to have been able to use the VPL to solve more exercises) correlates (0.607) with question Q13, revealing opinions concordant between considering the VPL as an added value and the willingness to use it.

It is worth mentioning the connection of question Q13 (the VPL is an added value for the teaching-learning process of programming) with several other questions, which is summarised in Table 48 where the values of the rho correlation coefficient of Spearman can be observed (Table 49).

Table 49 – Significant associations of question Q13 with others

		Spearman Coefficient (ρ)	p-value
Q13	Q4	0,442**	0,000
	Q5	0,689**	0,000
	Q7	0,436**	0,000
	Q8	0,607**	0,000
	Q10	0,418**	0,000
	Q11	0,462**	0,000
	Q14	0,505**	0,000

Looking at the data from Table 48 and Table 49 it is possible to observe that there are several questions with strong correlations common to Q4 and Q13, as illustrated in Figure 107.

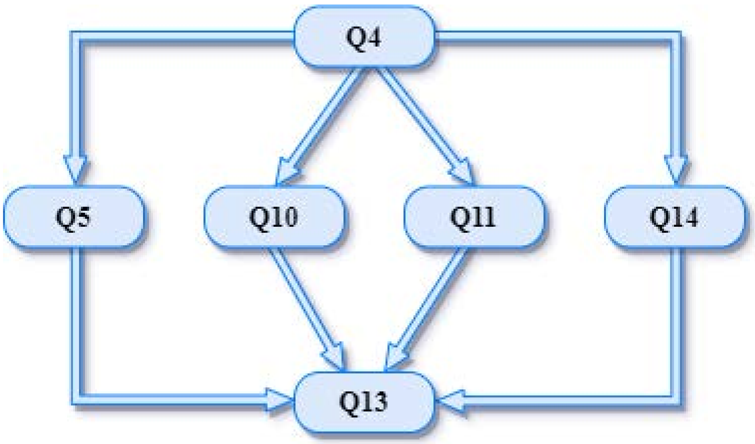


Figure 107 – Strong correlations common to questions Q4 and Q13

Questions Q5 (The use of VPL in the resolution of exercises was helpful to my learning process), Q10 (The possibility of resubmission and obtaining automatic classification is very useful), Q11 (The pre-defined tests were important in identifying deficiencies in my resolutions) and (Q14 (The experience was explained clearly) are directly linked to the achievement of the experiment, highlighting the significant impact of the usefulness of the demonstration exercise (question Q4). On the other hand, the issues mentioned also have a strong correlation with question Q13.

In conclusion, questions Q4 and Q13 can be said to be "anchor" questions, the former as a starting point for the achievement of the experiment, aggregating operational aspects, and Q13 from the standpoint of a global view of the process and the use of VPL.

6.4.7 Exploratory Factorial Analysis

Exploratory Factorial Analysis is one of the multivariate statistical techniques widely used in the statistical analysis of surveys (Hongyu, 2018). One of the measures that allows its suitability to be verified is the Cronbach alpha. This coefficient indicates the reliability degree of a survey, i.e. its internal consistency (Maroco & Garcia-Marques, 2006), and is expressed by a value between 0 and 1. According to Nunnaly (Nunnally, 1994), the instrument or test is considered reliable if it has a minimum Cronbach alpha of 0.7. According to Kline (Kline, 2000), a value of over 0.6 can already be considered acceptable as shown in Table 50.

Table 50 – Internal consistency according to Cronbach's alpha value

(Kline, 2000)

Cronbach's Alpha	Internal consistency
α ≥ 0,9	excellent
0,7 ≤ α < 0,9	good
0,6 ≤ α < 0,7	acceptable
0,5 ≤ α < 0,6	weak
α < 0,5	unacceptable

The following formula has been proposed for the determination of this coefficient:

$$\alpha = \frac{k}{(k-1)} \times \left[1 - \frac{\sum_{j=1}^{k} S_j^2}{S_T^2}\right]$$

where k corresponds to the number of items in the instrument, S_j^2 is the item variance and S_T^2 is the scale totals variance (Maroco & Garcia-Marques, 2006).

In the year 2017/2018 this analysis was carried out, with a calculated Cronbach alpha of 0.614, thus revealing a questionable internal consistency. For this reason, data from that year's surveys are not presented here and were not considered for the study's conclusions. This was also the reason why the one used in 2018/2019, based on that of the previous year, was revised and increased. This survey showed good internal consistency, as the Cronbach alpha obtained was 0.822 and can thus be considered reliable.

The Kaiser-Meyer-Olkin (KMO) measure and the Bartlett Sphericity Test (BTS) were used to check the quality of the correlations between variables, which are two of the commonly used assessment methods (Dziuban & Shirkey, 1974).

The KMO is an index that can vary between 0 and 1 and gives an indication of the sample suitability (Hongyu, 2018), according to the classification expressed in Table 51.

Table 51 – Suitability of the sample according to the KMO

(Hutcheson & Sofroniou, 1999)

KMO	Classification
KMO ≥ 0,9	excellent
0,8 ≤ ρ < 0,9	optimal
0,7 ≤ ρ < 0,8	good
0,5 ≤ ρ < 0,7	mediocre
KMO < 0,5	unacceptable

Its calculation is made by the following expression (Hongyu, 2018):

$$KMO = \frac{\sum_{j=1}^{p} \sum_{m=1,m\neq j}^{p} {r^2}_{jm}}{\sum_{j=1}^{p} \sum_{m=1,m\neq j}^{p} {r^2}_{jm} + \sum_{j=1}^{p} \sum_{m=1,m\neq j}^{p} {r^2}_{pjm}}$$

In the case of the 2018/2019 surveys, the KMO value obtained was 0.843, which is considered optimal, so the use of factor analysis applied to the data set obtained is appropriate.

Bartlett's sphericity test allows the evaluation of the general significance of all correlations in a data matrix, with significance levels under 0.05 indicating the suitability of factor analysis (Tabachnick & Fidell, 2006). Having obtained 681,304 for BTS, with a significance of 0,000, these values are indicative of the suitability of the factor analysis process for the data to be treated.

Thus, the factorial analysis was applied to the data obtained, and questions Q2 and Q12 were removed because they did not fit into any of the dimensions found. After further analysis, now free of the mentioned issues, three distinct dimensions were identified and categorised into three: Conducting the experiment, Using the VPL and Learning.

Table 52 - Results of the application of the Exploratory Factorial Analysis to the survey responses

Dimension	Question	Dimension		
		Conducting the experiment	Using the VPL	Learning
Conducting the experiment	Q15	0,853		
	Q14	0,849		
	Q4	0,686		
Using the VPL	Q6		0,722	
	Q8		0,69	
	Q5		0,607	
	Q13		0,586	
	Q7		0,515	
Learning	Q1			0,730
	Q9			0,659
	Q3			0,598
	Q10			0,581
	Q11			0,538
Cronbach's Alpha		0,782	0,800	0,676
Proprietary values		4,833	1,680	1,166
% of variance		37,176	12,925	8,972

Table 52 shows the identified dimensions, the questions associated with each of these dimensions, and a summary of the coefficients calculated by applying the Exploratory Factorial Analysis.

The three dimensions identified, corresponding to the conducting of the experiment, the use of the VPL in general and the learning process, are in line with the questions posed, as shown in Table 52, and presented below.

The issues related to the operational aspects of carrying out the experiment are:

- Q4 – The demonstration exercise was useful for getting to know the VPL
- Q14 – The experiment was explained clearly
- Q15 – Enough support was given for the effective use of the VPL

With regard to the VPL, to its general use and usefulness, the related issues are:

- Q5 – The use of VPL in the resolution of exercises was helpful for my learning process
- Q6 – I began by using the VPL but lost interest

- Q7 -The use of VPL is too complicated
- Q8 – I would have liked to have been able to use the VPL to solve more exercises
- Q13 – The VPL is an asset for the teaching-learning process of programming

As for the learning process based on the VPL, it is possible to assess it from answers to questions:

- Q1 – Programming is a difficult task
- Q3 – I would like to have tools to support the resolution of exercises outside the classroom
- Q9 – I would like to be able to use the VPL for individual assessment rather than the paper-based version
- Q10 – The possibility of resubmission and obtaining automatic classification is very useful
- Q11 – The pre-set tests were important in identifying shortcomings in my resolutions

Once the dimensions and questions for each one have been identified, the scores corresponding to the answers have been determined in order to establish a score per dimension, the procedure used being the one previously described in section 6.4.3.

Chart 22 - Score on the implementation of the experiment

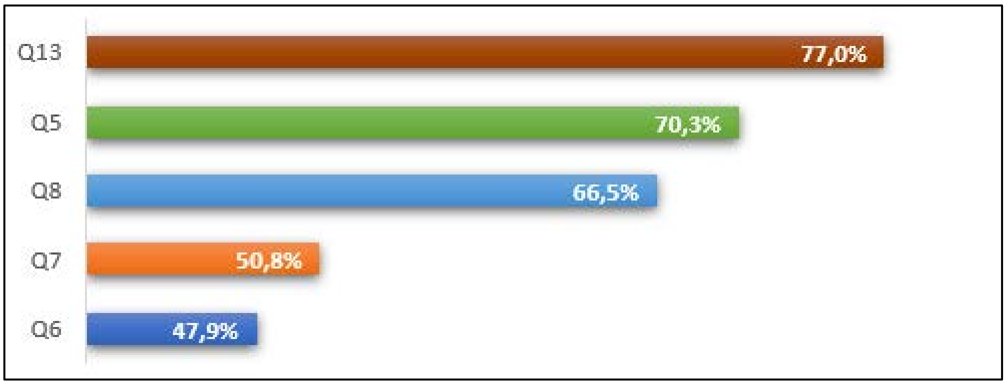


Chart 23 - Score on the use of the VPL

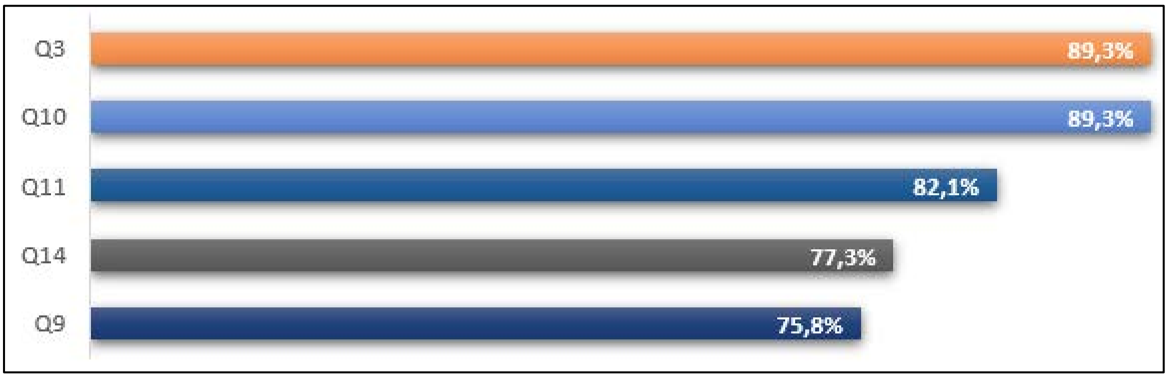


Chart 24 - Score on learning

From observing Chart 22, Chart 23 and Chart 24 it can be concluded that students consider the three dimensions to be important overall, although less relevance is evident in comparative terms in relation to the overall view and use of the VPL. With regard to the implementation of the experiment and learning, it is inferred that students assess the process as being effective, with each dimension having a minimum agreement level of over 75%.

6.5 RESULTS FOR TEACHERS

A survey was carried out among the teachers who collaborated on the experiment, in order to gather their opinion on the use of the VPL and the usefulness of this tool, as well as to obtain suggestions for improvement. The time each teacher uses to evaluate each exercise and the total time spent in class for that purpose were also queried.

Four teachers participated in each of the editions of the experiment, and in the second there were two teachers who had also participated in the first one. Thus, in total there were six teachers involved in the process, one of them being the author of this work who, despite having participated in all editions, refrained from answering the survey so as not to influence results. It should be noted that the data collected came from activities carried out in the years 2017/2018 and 2018/2019.

6.5.1 Description of the participating teachers

Chart 25 shows the distribution by age group of the teachers involved in the experiment, with the majority aged between 40 and 50.

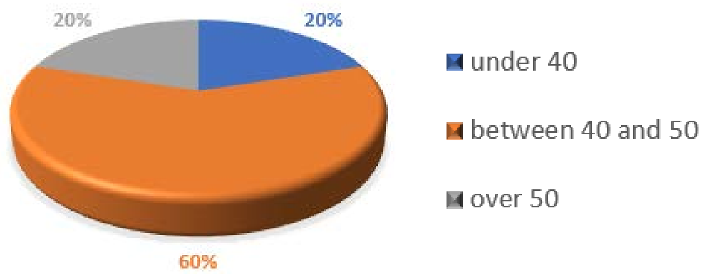


Chart 25 - Teachers' age group

With regard to basic training, although the overwhelming majority is in the IT field, there is some heterogeneity, as can be seen in Chart 26.

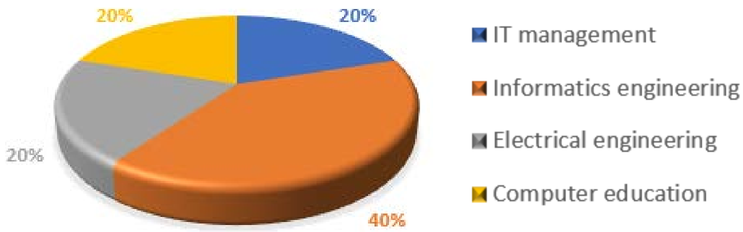


Chart 26 - Background training field

Teachers who have been involved in the use of the VPL have, overall, a considerable amount of teaching experience. The teacher with less teaching experience combines teaching with a professional activity in a business context, representing an added value in the link between the academic environment and the reality of companies.

Chart 27 presents the overall teaching experience.

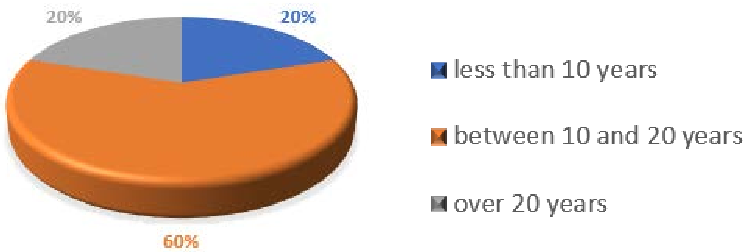


Chart 27 - Teachers' experience

Some of the teachers have taught in other contexts and/or areas, so their teaching experience in the context of algorithmics and programming is shorter than their overall experience (Chart 28).

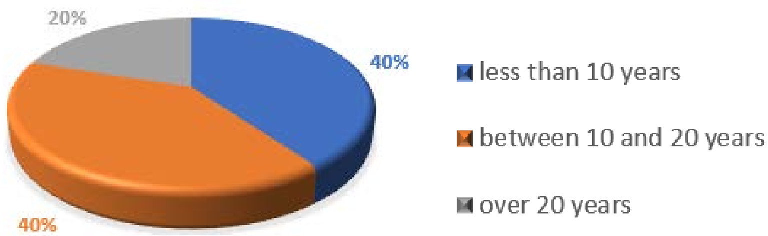


Chart 28 - Teaching experience in the context of algorithmics and programming

6.5.2 Results of teachers' surveys

For the reasons mentioned above only five answers were obtained and the results of this survey are therefore not statistically significant.

Despite this fact, it was decided to consider the answers regarding the time used in the analysis of exercises, since they were obtained from teachers with extensive experience in teaching the subject in question, most of them with several years of participation in the APROG curricular unit.

It is also noted that the times reported by each of the teachers (Chart 29) were very approximate, so they appear to be realistic and can be considered for analysis purposes.

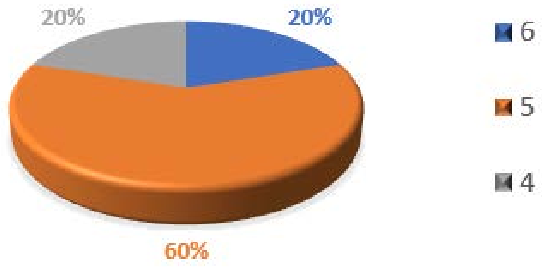


Chart 29 – Average time spent (in minutes) on validating each exercise

It should be noted that the mode and the mean obtained both correspond to five minutes.

Note that, on average, a PL class has 18 students, which, if working in pairs, will mean that each teacher will have to check the same exercise at least nine times in each class. Considering the average time of five minutes per exercise, for a universe of 13 classes (those who participated in the experiment in 2018/2019), the total average time needed to analyze an exercise would correspond to 585 minutes (13 classes x 9 exercises x 5 minutes).

It should be noted that one of the objectives of this study was to analyze the impact of the use of VPL on reducing the teacher's workload in the evaluation of assignments.

Each exercise may have to be checked more than once, because it may have syntax problems and its functionality cannot be checked, or, even if it works, because it does not fulfil the function for which it was designed. It may still work in some cases, but not in others because it does not meet all the requirements. Apart from this functional verification, it is important to perform a code analysis in order to verify the coding approach in terms of solution and its implementation. Thus, the time required to check a particular exercise can vary greatly depending on the initial resolution made by the student.

The empirical evidence and the opinion expressed by colleagues clearly indicated that the use of the VPL has significantly reduced the time dedicated to this activity.

Until an exercise used in APROG can be made available in the VPL, several tasks need to be performed, such as:

- analysis of the existing exercise;
- possible changes to the wording;
- coding in Java;
- defining and writing test cases;
- parametrizations;
- setting up the exercise in the VPL;
- functionality check;
- test coverage check;
- submission test with a specific user with student permissions only.

In order to try to estimate the potential time gains with the use of the VPL, the approximate duration of the preparation of the exercises to be used in the VPL was accounted for throughout the process. This record was made by the author of the study and analyzed and corroborated by the colleague who was involved in all editions of the experiment and who followed the process closely.

For the exercises used, an estimated average preparation time for each exercise was calculated to be 120 minutes. It is worth remembering that, in a universe of 13 classes, a verification time of each exercise of 585 minutes has been pointed out, the usual number of classes in APROG is 17, which would correspond to a time of 756 minutes.

Of course, the VPL will not release the teacher from time spent on analysis and feedback of the exercises. However, the time needed will certainly be shorter and at a stage when several of the problems should already have been detected by the VPL and corrected by students.

The questionnaire also included a section on the use of VPL and the teaching-learning process, with the questions from Table 53.

Table 53 - Questions for teachers

Question	Description
Q1	I would have liked to have been able to use the VPL for the resolution of more exercises
Q2	It would be useful to use the VPL for individual assessment in place of the paper-based version
Q3	The VPL can contribute to the uniformity of the subject's grades, considering the volume of students, work, exercises and different assessments carried out
Q4	The VPL can decrease the time the teacher uses in the teaching/learning and assessment process
Q5	Compared to other school years, the use of the VPL has had a positive impact on learning
Q6	The feedback given automatically and immediately by the LPV is useful for the student
Q7	The VPL is an added value to the teaching-learning process of programming

The results of the answers to these questions are aggregated in Chart 30, allowing us to observe that the majority of the questions were fully or partially agreed with by the respondents.

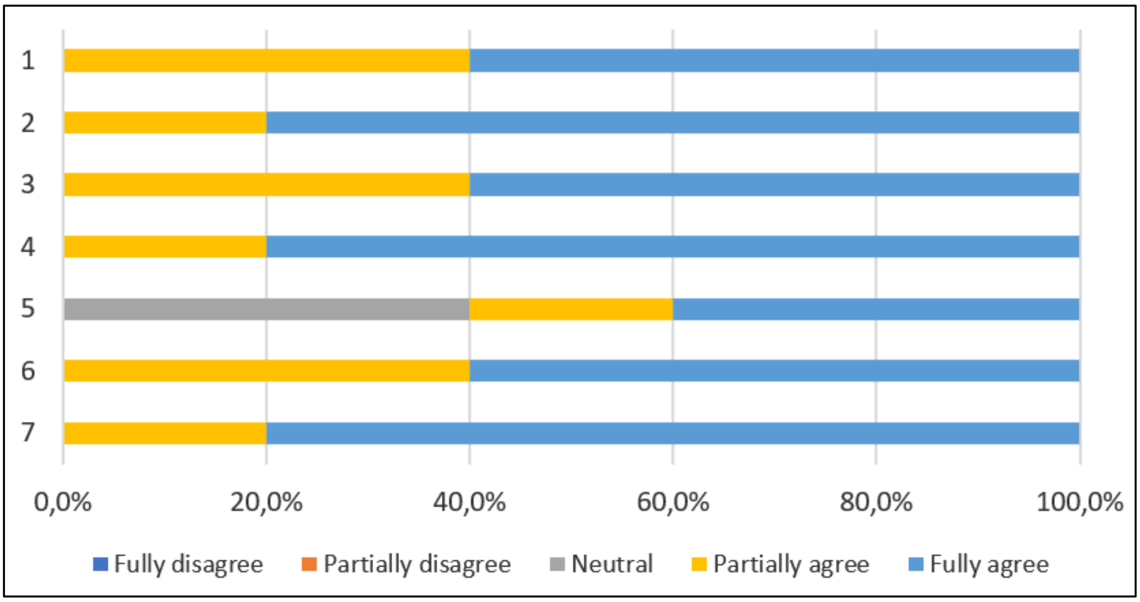


Chart 30 - Teachers' responses to the opinion survey

As can be seen, the general opinion is very much in agreement with the statements and in favour of the use of VPL, the only exception being the answer to question Q5, concerning the impact on learning (Chart 31).

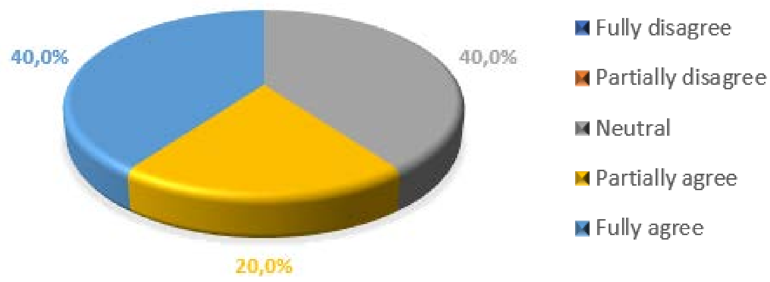


Chart 31 – Positive impact of the VPL on learning compared to other years

Two of the teachers fully agree and one partially agrees that the VPL has had a positive impact on learning compared to previous years. However, the remainder report that they did not find that the VPL had an impact (positive or negative) on learning.

Using the same conversion of the original scale to numerical values (Table 27) mentioned in section 6.4.3, the mean, median, standard deviation and coefficient of variation of each of the questions were calculated (Table 54).

Table 54 – Statistical figures on teachers' replies

Question	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Mean	4,60	4,80	4,60	4,80	4,00	4,60	4,80
Median	5,00	5,00	5,00	5,00	4,00	5,00	5,00
Standard Deviation	0,55	0,45	0,55	0,45	1,00	0,55	0,45
Coefficient of variation	12%	9%	12%	9%	25%	12%	9%

The number of respondents is very limited, the responses being very concordant as can be seen from the standard deviation values and the low coefficients of variation.

6.5.3 Nonparametric correlations - survey to teachers

Also in relation to the surveys carried out on teachers, the Spearman correlation matrix (Table 55) was determined, in order to identify possible associations between the questions, using Microsoft Excel and IBM SPSS Statistics (version 25).

Table 55 - Spearman's Matrix of Non-Parametric Correlations

		Q1	Q2	Q3	Q4	Q5	Q6
Q2	ρ sig.	0,612 (0,272)					
Q3	ρ sig.	0,167 (0,789)	-0,408 (0,495)				
Q4	ρ sig.	-0,408 (0,495)	-0,250 (0,685)	0,612 (0,272)			
Q5	ρ sig.	0,000 (1,000)	0,559 (0,327)	0,000 (1,000)	0,559 (0,327)		
Q6	ρ sig.	0,167 (0,789)	0,612 (0,272)	0,167 (0,789)	0,612 (0,272)	0,379** (0,030)	
Q7	ρ sig.	-0,408 (0,495)	-0,250 (0,685)	0,612 (0,272)	1,000** (0,000)	0,559 (0,327)	0,612 (0,272)

* Correlation is significant at the 0.05 level (2-tailed).

** Correlation is significant at the 0.01 level (2-tailed).

Based on the matrix data, it can be observed that questions Q5 and Q6 show an almost perfect association and that questions Q4 and Q7 show the same levels of agreement.

The remaining issues do not show significant associations, which reflects a certain independence in teachers' perceptions.

It should be noted that the teacher surveys were not anonymous, requiring the use of credentials to prove the authenticity of the respondent's identity. However, their responses were voluntary and were not discussed in advance so as not to influence their judgement. But, knowing the identity of the respondents, it was possible to obtain further clarification of their responses, in particular from those who expressed an opinion on the neutrality of VPL in learning.

It was thus possible to ascertain that the choice made was based on the results obtained among the students of the year in question and of the previous year and on some difficulties experienced by some students in validating the exercises due to the strictness of output.

It should be noted that two of the colleagues concerned only participated in the experiment in the first year (2017/2018) so their opinion was based only on the first edition which served mainly as an exploratory phase for the following edition.

It should be reiterated that this was a pilot experiment and that, for the reasons explained above, the VPL was not used massively, but was applied to only six exercises. In addition, there are several other factors that have an influence on learning, the classes being typically very heterogeneous and different from each other.

7 CONCLUSIONS AND FUTURE WORK

"An experience is never a failure, as it always demonstrates something."

Thomas Edison

This last chapter is intended to provide a reflection on the work carried out in this study and its main results and contributions.

Thus, the results of the research are presented with respect to the objectives achieved, as well as the contributions to the teaching-learning process of programming.

Possible future developments of the work undertaken are also identified and the final considerations presented.

7.1 ACHIEVED OBJECTIVES

The study presented here had as a main objective to investigate the potential of integrating the VPL in the teaching-learning process of programming. This analysis was carried out with potential advantages for those involved in the process in mind: teachers and students.

Depending on the problem identified, experiments were carried out in a real teaching context, from which data were obtained whose analysis made it possible to assess the achievement of the objectives of the study.

The following specific objectives have been identified to assess the potential of the VPL:

- To verify the feasibility and usefulness of using the VPL in the teaching of curricular units in higher education;
- Check the compatibility of the use of the VPL with the eduScrum methodology;
- Check whether it is possible to make student learning more effective and autonomous with VPL;
- To investigate the impact of the use of VPL in reducing the workload of the teacher in the evaluation process;
- Analyse whether the use of VPL can contribute to a quicker and fairer evaluation by standardising the evaluation mode and the use of anti-plagiarism tools;
- Justify extending the use of the VPL, or other similar tool with the same purpose, to all students of APROG, as well as to other course units and/or other courses.

From the results presented in the previous chapter, namely from the mostly positive assessments of the different actors, it is possible to infer that the VPL can be useful in the teaching-learning process of programming in higher education.

The overall analysis of the results leads to the conclusion that the profile of the respondents shows a strong disposition to the use of technologies, associated with the use of tools that facilitate autonomous study, this being an expected result in the context of a computer engineering course. Regarding the perception of difficulty of programming activity (question Q1 Table 25) there is a broad diversity of opinions, with statistically significant differences regarding gender, age group and previous experience in algorithmics and programming. It can be observed that previous programming experience leads to considering the task of programming as less difficult (Table 36).

This is a conclusion that may prove important and may help to demystify the complexity associated with the task of programming, evidenced by the fact that students who perceive the task of programming as being difficult place more value on the use of the VPL. As these are presumably the most difficult to evolve, they may have an ally in the VPL to help them develop their codification skills.

As for the use of VPL, the vast majority of students showed interest, skill and enthusiasm in its use, observed directly by the teachers and corroborated by the answers to the survey, in particular questions Q6, Q7 and Q10, as can be seen in section 6.4.3. The general opinion of the students is frankly favourable regarding the teaching-learning process supported by VPL, as evidenced by the answers to questions Q5 and Q13, and the experience has been appreciated as evidenced by the score associated with that dimension (Chart 22).

Having the experiment in APROG based on the eduScrum model, the use of the VPL was found to be compatible with this methodology. Throughout the lessons it was possible to observe that the students, despite working in groups, made their submissions individually, but in case of failure they supported each other, discussing the reasons and possible solutions.

Increased autonomy was also an aspect to be examined. It was evident from the answers obtained and the observations made that one aspect that was highly valued by the students was the automatic feedback based on the pre-defined tests, which allowed work to be carried out at any time and place, without the presence of the teacher. This is evidenced by the statistically significant differences between the two groups considered in the number of exercises submitted.

The impact of the use of the VPL, measured by means of the questionnaire, allowed the identification, through non-parametric correlations and exploratory factorial analysis, of three dimensions that seem to be markedly important, namely: the conducting of the experiment, the use of the VPL and learning (Table 52 and Figure 108).

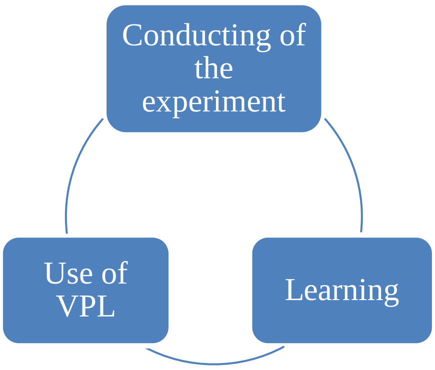


Figure 108 – Dimensions associated with student surveys

Regarding the results of Spearman's non-parametric correlations, significant associations were highlighted among several issues, those mentioned in Figure 107, from a first familiarization with the system to a macro and global view of the VPL.

The fact that there are several evaluation moments, the excessive workload of teacher evaluation and the teacher's lack of availability for more effective student support was one of the problems identified. The results of the use of the VPL point to an effective means of supporting the teacher. Nevertheless, the preparation of exercises for the VPL was fairly demanding and time consuming, involving the definition of test cases, parameterisation and possible adjustments, and the preparation time for each exercise can be relatively long. However, this preparation is carried out beforehand and once only, regardless of the number of students. Thus, for a reality such as APROG, with hundreds of students, the time balance is clearly positive, as was seen in section 6.5.

Plagiarism detection functionality was also one of the requirements defined for the selection of the tool to adopt. This option was intended to make it possible to analyse similarities in line with the trend in fighting academic fraud. To this end some exploratory tests were carried out which proved unpromising for APROG. The fact that it is a context of initiation to programming with very simple exercises, promotes little variability in the solution proposals. In terms of use for summative assessment purposes, the VPL also proved difficult to apply in the context of APROG, as mentioned at the end of section 5.4.5.

A further objective was to promote the use of VPL in APROG and other contexts. As a result of the study presented here, a code evaluation tool was adopted in APROG in the school year 2019/2020, being used by all students and for all exercises made available for resolution. By decision of the new APROG RUC, Mooshak was adopted and not the VPL, due to the fact that the RUC already knew the former and had experience in its use, as mentioned in section 5.4.6. In addition, DEI is planning to organise programming competitions where Mooshak will be used, with the aim of increasing the number of participants in the competition. The discussion of this study has contributed greatly to this new reality.

In addition to the proposed objectives, a number of facts have emerged from the experience and which deserve to be highlighted:

- The use of the VPL has raised the need to take into account some technical aspects that would otherwise be underestimated or even totally ignored, thereby enriching the process;
- The preparation of the exercises required greater attention from teachers to clarify and verify their functioning in the VPL. This has led to more careful and tested exercises, significantly improving their quality.

In the course of the study an opportunity was also identified to improve the process by including means of coding style verification. As such, although not an initial objective, functionalities have been added to the VPL that may represent an important added value for student learning.

Overall, the objectives of this work were achieved, and it can be concluded that the teaching-learning model of initiation to programming with the support of the NPV has advantages for students and teachers, compared to the traditional model.

7.2 CONTRIBUTIONS

The main contribution of this study is to validate that a tool such as the VPL can be rather useful and contribute to the teaching of programming by perfectly adapting to the eduScrum model.

At the same time, it has contributed to a broadening of the horizons of teachers and students by introducing them to teaching tools and methodologies that can contribute to new paradigms in the teaching of programming.

This doctoral work is considered to have contributed to the development of the teaching-learning process of programming in the following areas:

a) identifying a concrete problem in teaching introduction to programming, its causes and consequences;
b) identifying, analysing and characterising automatic evaluation tools to assist in teaching programming;
c) presenting a proposal for a solution to the problem identified;

d) carrying out a study using a tool, tested in real class context;
e) deepening the functionalities of the tool used by enhancing evaluation and the possibility of inducing good code writing practices;
f) implementing a system with a repository in the cloud and with version control, facilitating the use and development of new features;
g) reporting of the experiment and sharing of results, contributing to the intensification of the use of automatic code evaluation tools.

7.3 PUBLICATIONS, PARTICIPATION IN EVENTS AND OTHER ACTIONS

In a work of this nature and dimension, several other actions were developed in addition to the activities within the scope of the experience.

During the development of this work some publications and presentations were made on the VPL and on the experiment carried out.

7.3.1 Publications

The study presented and discussed here led to the production of some publications, with a list of the articles published during the period of the study and in direct relation to it being presented below.

- Marílio Cardoso, Rui Marques, António Vieira de Castro, Álvaro Rocha, “Using Virtual Programming Lab to improve learning programming: The case of APROG”, published in the journal “Expert Systems” with JCR impact factor, 2020, ISSN: 1468-0394.
- Marílio Cardoso, Rui Marques, António Vieira de Castro, “Promoting success in beginner programming students with Virtual Programming Lab”, in book of abstracts from CASHE, Conference Academic Success in Higher Education (pp. 28-29), presented at CASHE, Porto, 2019. ISBN: 978-989-54236-4-4.
- Marílio Cardoso, Rui Marques, António Vieira de Castro, “Promoting coding best-practices by extending Moodle’s VPL”, in book of abstracts from CASHE, Conference Academic Success in Higher Education (pp. 38-39), presented at CASHE, Porto, 2019. ISBN: 978-989-54236-4-4.
- Marílio Cardoso, António Vieira de Castro, Rosa Barroso, Álvaro Rocha, Rui Marques, “Introducing VPL on a programming learning process”, in proceedings from EDULEARN18, 10th International Conference on Education and New Learning Technologies (pp. 8499-8508), presented at EDULEARN 2018, Palm of Majorca, Spain. ISBN: 978-84-09-02709-5, ISSN: 2340-1117.
- Marílio Cardoso, António Vieira de Castro, Álvaro Rocha, “Integration of virtual programming lab in a process of teaching programming EduScrum based", presented at 13th Iberian Conference on Information Systems and Technologies (CISTI), Doctoral Symposium, Cáceres, Spain, ISBN: 978-989-98434-8-6, doi:10.23919/CISTI.2018.8399261.
- Marílio Cardoso, Rosa Barroso, António Vieira de Castro, Álvaro Rocha, “Virtual Programming Labs in the computer programming learning process, preparing a case study”, in proceedings from EDULEARN17, 9th International Conference on Education and New Learning Technologies (pp. 7146-7155), presented at EDULEARN 2017, Barcelona, Spain. ISBN: 978-84-697-3777-4, ISSN: 2340-1117.

7.3.2 Participation in conferences and events

The author of this study has participated in some events where he presented and promoted this study.

7.3.2.1 CNaPPES.19

The author participated in the 6th National Congress of Pedagogical Practices in Higher Education, CNaPPES.19, which took place at the School of Agrarian Higher Education of the Polytechnic Institute of Santarem on July 11 and 12, 2019.

At this event the communication "The teaching of programming and the *Virtual Programming Lab*" was presented, in a session on the theme "Technologies in the classroom and projects".

7.3.2.2 CASHE 2019

CASHE, (*Conference Academic Success in Higher Education*), was held at the Instituto Superior de Engenharia do Porto, on February 14 and 15, 2019.

In this event, the author participated in the workshop "*Automative student assessment*" and presented the article "Promoting success in beginner programming students with Virtual Programming Lab".

7.3.2.3 EDULEARN18

EDULEARN18,*10th International Conference on Education and New Learning Technologies*, was held in Palma de Mallorca, Spain, from July 2-4, 2018.

The article "*Introducing VPL on a programming learning process*" was published in the conference proceedings book and its presentation was given by virtual participation of the author.

7.3.2.4 CISTI'2018

The author presented the article "*Integration of Virtual Programming Lab in a process of teaching programming EduScrum based*", at the Doctoral Symposium at the 13th Iberian Conference on Information Systems and Technologies - CISTI'2018, having also been moderator of the session Information Technologies in Education.

The conference was held in Caceres, Spain, from 13 to 16 June 2018.

7.3.2.5 EDULEARN17

At EDULEARN17, the 9th International Conference on Education and New Learning Technologies, which took place in Barcelona, Spain, from 3 - 5 July 2017, the paper "Virtual Programming Labs in the computer learning process, preparing a case study" was presented (Figure 109).

In this conference, the author was also moderator of the session "Coding and Programming in Schools".

Figure 109 – Participation in EDULEARN17

7.3.3 Other actions

In addition to the activities already mentioned, there have been other opportunities to publicize the experience that has supported the study presented here, as well as to undertake other actions within the scope of the same study.

7.3.3.1 Presentations to teachers

One such opportunity arose during a training course for secondary school teachers called "Digital Teaching Skills". In this training, several resources were presented and tested, and the author made a presentation on the VPL and its use.

In an organization of the e-Learning and Pedagogical Innovation Unit of the Polytechnic of Porto (EIPP), the tertulia "Relação Pedagógica e Comunicação" (Pedagogical Relationship and Communication) was held. The event was attended by teachers from various schools of the Polytechnic of Porto, allowing for the exchange of experiences, and the author of this work shared the description of the experience made using the VPL with colleagues who attended.

Also, as part of the EIPP and its training plan, the author has been invited to participate in the Moodle 4.0 course for teachers on advanced use of Moodle. At that event he was given the opportunity to present the VPL, explain how it has been used in this work and briefly demonstrate its operation (Figure 110).

Figure 110 - Presentation of VPL in the "Moodle 4.0" course

7.3.3.2 Development of a course for secondary school teachers

The tools for automatic code evaluation are still rather unknown, it being understood that their use should be more widespread and intensified. In this sense, an introduction course to Java has been designed for secondary school teachers, with the use of automatic evaluation tools in their program, in particular the VPL and Mooshak.

The course was submitted for accreditation by the Scientific-Pedagogical Council of Continuing Education, and received a favourable ruling on 2018-10-25 and was awarded the CCPFC/ACC-101425/18 accreditation registration (Annex I). The author is one of the course applicants and one of its trainers. The course is being launched at DEI/ISEP for the first edition.

7.3.3.3 Implementation of VPL in the European Up2U project

The author is part of the Portuguese team of the European project Up2U[111] (Up to University). This project aims to bridge the gap between secondary and higher education by promoting the integration of formal and informal learning scenarios and fostering the use of technology and methodology that students are likely to encounter in higher education.

Within this project an ecosystem was implemented, based on Moodle, which was made available to European secondary schools. To add to the ecosystem, besides the base being Moodle, many tools and functionalities were added and, in that context, we were invited to install the VPL plugin in that ecosystem in order to provide secondary school teachers, of the programming areas, a contact with this Virtual Programming Laboratory and promote its use. The VPL was installed in the Up2U ecosystem and the creation of such activities is currently available.

7.4 FURTHER WORK

Throughout this work several situations and realities arose that imposed some changes to the initial plan. In some cases, these unforeseen events resulted in difficulties, which were overcome, and which, taken as opportunities, led to an increase in the quality of work, making new developments possible.

This being a work with defined objectives and deadlines, it would not be desirable, nor possible, to proceed with other developments beyond those presented here. However, given their relevance and the added value they may represent, they are listed here as a proposal for future work:

- Refine the developed scripts in order to render more flexible *output* and improve the analysis of code writing;
- Extend the use of VPL to other LEI curriculum units (for example: Programming Paradigms - PPROG and Databases - BDDAD) and other DEI-ISEP courses;
- Creation of a template questionnaire to serve as a validation tool for the VPL;
- Writing of scientific papers related to the present study and its presentation in conferences and other events related to the subject.

At the same time, the following actions can also be considered:

- Development of a user's manual for the VPL, with references to the added improvements and limitations and/or constraints of the tool;
- Development of a Frequently Asked Questions document (FAQ - Frequently Asked Questions) on the use of VPL;
- Making a video explaining the use of VPL by teachers, including the settings and definition of test cases;
- Making a video explaining the use of VPL by students, examples of submissions and feedback;

[111] https://up2university.eu/

- Further experiences in the use of the tool for summative evaluation, with further use of the anti-plagiarism component;
- Gathering information on the use of the tool with respect to frequency, time and duration of access, to try to draw up a usage profile;
- Carrying out periodic surveys on the use of the VPL in order to identify difficulties and enhance their resolution;
- Holding of a dissemination and demonstration session for DEI teachers.

7.5 CLOSING REMARKS

The work presented here represents the culmination of a study and research that is expected to be a valid contribution in the scientific and pedagogical aspects in the area of Computer Science, and in particular, in the teaching-learning process of programming.

It is hoped that it can be replicated and deepened not only at ISEP, but also in other schools and other contexts, in the initial approach to teaching programming, assisting students and teachers.

The potential of this type of tools is beginning to be revealed in their use by IT companies for the global selection process. Currently, there are companies selecting employees using automatic code validation models, so providing this approach to students may help them feel more comfortable with this type of process of possible hiring.

REFERENCES

ACM/IEEE. (2013). *Computer Science Curricula 2013 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. Final Report.* New York, USA, Undergraduate Degree Programs in Computer Science." Final Report, 2013. DOI: 10.1145/2534860 Web link: http://dx.doi.org/10.1145/2534860: ACM. doi:10.1145/2534860

Adams, J. C., Brainerd, W. S., Hendrickson, R. A., Maine, R. E., Martin, J. T., & Smith, B. T. (2009). *The Fortran 2003 Handbook: The Complete Syntax, Features and Procedures.* Springer-Verlag. doi:10.1007/978-1-84628-746-6

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. *39th EUROMICRO SEAA Conference*, (pp. 9-16).

Al Shamsi, F., & Elnagar, A. (2012). An Intelligent Assessment Tool for Students' Java Submissions in Introductory Programming Courses. *Journal of Intelligent Learning Systems and Applications, 4*, 59-69. doi:10.4236/jilsa.2012.41006

Al-Ajlan, A., & Zedan, H. (2008). Why Moodle. *FTDCS '08 Proceedings of the 2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems* (pp. 58-64). Washington: IEEE Computer Society . doi:10.1109/FTDCS.2008.22

Alammary, A., Sheard, J., & Carbone, A. (2014). Blended learning in higher education: Three different design approaches. *Australasian Journal of Educational Technology, 30*(4). doi:doi.org/10.14742/ajet.693

Ala-Mutka, K. M. (2005). A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education, 15*(2), 83-102. doi:10.1080/08993400500150747

Aleksic, V., & Ivanovic, M. (2016). Introductory Programming Subject in European Higher Education. *Informatics in Education, 15,* 163-182. doi:10.15388/infedu.2016.09

Al-Emran, M., ElSherif, H. M., & Shaalan, K. (2016). Investigating attitudes towards the use of mobile learning in higher education. *Computers in Human Behavior, 56*, 93-102.

Allevato, A., & Edwards, S. H. (2012). RoboLIFT: Engaging CS2 Students with Testable, Automatically Evaluated Android Applications. *Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE '12)* (pp. 547-552). Raleigh, North Carolina, USA: ACM. doi:10.1145/2157136.2157293

Almeida, R. S. (2013). *Aprendendo Algoritmo com VisuAlg.* Editora Ciência Moderna.

ANSI. (1989). *Programming Language C: American National Standard for Information Systems : ANSI X3.159-1989.* American National Standard for Information Systems.

Arnold, K., Gosling, J., & Holmes, D. (2015). *The Java Programming Language* (fourth ed.). Addison Wesley Professional.

ASME. (1947). *ASME Standard: Operation And Flow Process Charts.* New York: American Society of Mechanical Engineers.

Atzori, L., Morabito, G., & Iera, A. (10 de 2010). The Internet of Things: A survey. *Computer Networks, 54*(15), pp. 2787-2805.

Auffarth, B., López-Sánchez, M., i Miralles, J. C., & Puig, A. (2008). System for automated assistance in correction of programming exercises (SAC). *Proceedings of V International Congress University Teaching and Innovation (CIDUI 2008)*, (pp. 104-113). Lleida, España.

Austin, M. J., & Brown, L. D. (1999). Internet Plagiarism: Developing Strategies to Curb Student Academic Dishonesty. *Higher Education Research and Development, 32*(3), pp. 369-380. doi:10.1080/07294360.2012.687362

Awad, M. A. (2005). A Comparison between Agile and Traditional Software Development Methodologies. *Submitted as partial fulfilment of the requirements for the Honours Programme of the School of Computer Science and software Engineering, University of Western Australia*.

Backus, J. W. (1959). The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. *Proceedings of the International Conference on Information Processing* (pp. 125-132). UNESCO.

Backus, J. W. (1978). The history of FORTRAN I, II, and III. *SIGPLAN Notices, 13*(8), 165-180. doi:http://doi.acm.org/10.1145/960118.808380

Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perils, A. J., . . . Woodger, M. (1963). Revised Report on the Algorithmic Language Algol 60. *Communications of the ACM, 6*(1), 1-17. doi:10.1145/366193.366201

Badri, M., Badri, L., & William, F. (2016). Source and Test Code Size Prediction - A Comparison between Use Case Metrics and Objective Class Points. *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering* (pp. 172-180). Rome, Italy: SCITEPRESS - Science and Technology Publications, Lda. doi:10.5220/0005857601720180

Bates, A. W. (2015). *Teaching in a Digital Age.* Obtido de https://irl.umsl.edu/oer/6/

Beck, K., & Andres, C. (2004). *Extreme Programming Explained. Embrace Change.* Boston, Massachusetts, USA: Addison Wesley.

Bell, P., & Beer, B. (2015). *Introducing GitHub: A Non-Technical Guide.* Sebastopol, California, USA: O’Reilly Media, Inc.

Bennedsen, J., Caspersen, M. E., & Kölling, M. (2008). Reflections on the teaching of programming. Berlin: Springer-Verlag.

Bers, M. U., Flannery, L. P., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers and Education, 72*, 145–157. doi:10.1016/j.compedu.2013.10.020

Bilabila, A. M. (2017). CompAlg – Ferramenta de Ensino e Aprendizagem da Lógica de Programação. *Tese de Mestrado*. Faculdade de Ciências da Universidade do Porto.

Blumenstein, M., Green, S., Nguyen, A., & Muthukkumarasamy, V. (2004). GAME: A Generic Automated Marking Environment for Programming Assessment. *Proceedings ITCC 2004 International Conference on Information Technology: Coding and Computing. 2*, pp. 212-216. IEEE. doi:10.1109/ITCC.2004.1286454

Bransford, J. D., & Stein, B. S. (1984). *The IDEAL problem solver: A guide to improving thinking, learning and criativity* (English Language edition ed.). (W. H. Freeman, Ed.) W. H. Freeman and Company.

Brečko, B. N., & Carstens, R. (2006). Paper survey versus web survey - do they provide comparable results? *Proceedings of the 2nd IEA International Research Conference.* Washington.

Brezina, C. (2006). *Al-Khwarizmi: The inventor of algebra.* The Rosen Publishing Group.

Brooks, F. P. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer, 20*(4), pp. 10-19.

Bullen, M., Morgan, T., & Qayyum, A. (2011). Digital Learners in Higher Education: Generation is Not the Issue. (C. N. Education, Ed.) *Canadian Journal of Learning and Technology, 37*(1).

Buono, S. A. (2005). *C# and Game Programming: A Beginner's Guide* (second ed.). A K Peters/CRC Press.

Cairns, L., & Malloch, M. (2017). Computers in Education: The Impact on Schools and Classrooms. Em R. Maclean, *Life in Schools and Classrooms: Past, Present and Future* (Vol. 38, pp. 603-617). Singapore: Springer. doi:10.1007/978-981-10-3654-5_36

Caiza, J. C., & Álamo, J. M. (2013). Programming Assignments Automatic Grading: Review of Tools and Implementations. *7th International Technology, Education and Development Conference (INTED2013)*, (pp. 5691-5700). Valencia.

Campos, C. P., & Ferreira, C. E. (2004). BOCA: Um sistema de apoio para competições de programação. *Anais do Congresso da SBC - Workshop de Educação em Computação.*

Campos, E., & Ramos, S. (2013). A Motivação dos Estudantes no Ensino Superior: Um Estudo Comparativo. *Interações: Sociedade e as novas modernidades*(21). Obtido de https://www.interacoes-ismt.com/index.php/revista/article/view/324

Cardoso, J. d. (2017). Avaliação Automática de Programas em Contexto de E-learning. *Tese de Mestrado*. Faculdade de Engenharia da Universidade do Porto.

Cardoso, M., Barroso, R., Castro, A. V., & Rocha, Á. (2017). Virtual Programming Labs in the computer pogramming learning process, preparing a case study. *EDULEARN17 Proceedings, pp. 7146-7155.*

Cardoso, M., Castro, A. V., & Rocha, Á. (2018). Integration of Virtual Programming Lab in a process of teaching programming EduScrum based. *2018 13th Iberian Conference on Information Systems and Technologies (CISTI).* Cáceres, Spain.

Carnevale, A. P., Smith, N., & Melton, M. (2011). *STEM: Science, Technology, Engineering, and Mathematics.* Georgetown University Center on Education and the Workforce, Washington DC, USA. Obtido de cew.georgetown.edu/STEM

Carvalho, A. A. (2008). Os LMS no Apoio ao ensino Presencial: dos conteúdos às interacções. *Revista Portuguesa de Pedagogia, 42*(2), pp. 101-122.

Carvalho, A., Areal, N., & Silva, J. (2011). Students' perceptions of Blackboard and Moodle in a Portuguese university. *British Journal of Educational Technology, 42*(5). doi:10.1111/j.1467-8535.2010.01097.x

Castagnetto, J. M., Rawat, H., Schumann, S., Scollo, C., & Veliath, D. (1999). *Professional PHP Programming.* Wrox Press Inc.

Castro, A. V. (2005). Tecnologias Multimédia na Auto-Aprendizagem de Lógica e Linguagens de Programação. *Tese de Mestrado*. Faculdade de Engenharia da Universidade do Porto.

Castro, A. V. (2012). Metodologia para preservação e partilha de conhecimento para a área da saúde. *Tese de Doutoramento*. Faculdade de Engenharia da Universidade do Porto.

Castro, J. P., Perez, M. Á., Regueras, L. M., & Verdú, M. J. (2010). *EduJudge system handbook. How to organize programming competitions in Moodle courses.* Madrid, Espanha: Sello Editorial SL.

Chalk, B., & Fraser, K. (2006). A Survey on the teaching of introductory programming in Higher Education. *Proceedings of the 10th Java & the Internet in the Computing Curriculum Conference (JICC10).* Thomson Publishers.

Cheang, B., Kurnia, A., Lim, A., & Oon, W.-C. (2003). On Automated Grading of Programming Assignments in an Academic Institution. *Computers & Education, 41*(2), pp. 121-131.

Chiavenato, I. (2003). *Introdução à teoria geral da administração* (7ª ed.). Rio de Janeiro: Elsevier.

Christian, M., & Trivedi, B. (2016). A Comparison of Existing Tools for Evaluation of Programming Exercises. *Proceedings of the Second International Conference on*

Information and Communication Technology for Competitive Strategies (ICTCS'16). Udaipur, India: ACM. doi:10.1145/2905055.2905350

Churchill, D., Jie, L., Chiu, T. K., & Fox, B. (2016). *Mobile learning design: Theories and application.* Springer.

Clifton, J. (2010). A Simple Judging System for the ACM Programming Contest. *43rd Annual Midwest Instruction and Computing Symposium 2010 (MICS 2010)*, (pp. 400-411). Eau Claire, Wisconsin, USA. Obtido de http://pc2.ecs.csus.edu/

Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams.* Pearson Education.

Cohn, M. (2006). *Agile Estimating and Planning.* Prentice Hall.

Cohn, M. (2013). *Succeding With Agile: Software Development Using Scrum.* Addison-Wesley Professional.

Cole, J., & Foster, H. (2007). *Using Moodle: Teaching with the Popular Open Source Course Management System* (second edition ed.). O'Reilly Media.

Collins, A., & Halverson, R. (2009). *Rethinking education in the age of technology: The digital revolution and schooling in America.* New York, USA: Teachers College Press.

Colmerauer, A., & Roussel, P. (1996). The Birth of Prolog. Em T. J. Bergin Jr., & R. G. Gibson Jr, *History of programming languages---II* (pp. 331-367). New York, USA: ACM. doi:10.1145/234286.1057820

Combéfis, S., & de Saint-Marcq, V. l. (2012). Teaching programming and algorithm design with Pythia, a web-based learning platform. *Olympiads in Informatics*, (pp. 31-43).

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Third Edition ed.). The MIT Press.

Correia, H. P. (2017). Avaliação de diagramas no Mooshak 2.0. *Tese de Mestrado*. Faculadade de Ciências da Universidade do Porto. doi:https://repositorio-aberto.up.pt/handle/10216/108497

Crittenden, V. L., Hanna, R. C., & Peterson, R. A. (2009). The cheating culture: A global societal phenomenon. *Business Horizons*, pp. 337-346. Retrieved from https://doi.org/10.1016/j.bushor.2009.02.004

Crompton , H. (2013). A historical overview of mobile learning: Toward learner-centered education. Em Z. L. Berge, & L. Y. Muilenburg (Edits.), *Handbook of mobile learning* (pp. 3-14). New York: Routledge.

Cruz, A. M., Cerqueira, N., & Gradíssimo, O. (2009). Portugol Viana - Manual da Linguagem.

Dalmoro, M., & Vieira, K. M. (2013). Dilemas na construção de escalas Tipo Likert: o número de itens e a disposição influenciam nos resultados? *Revista Gestão Organizacional, 6*(3), pp. 161-174. Obtido de http://www.spell.org.br/documentos/buscaredicao/periodico/revista-gestao-organizacional/idedicao/3221

Daly, C. (1999). RoboProf and an Introductory Computer Programming Course. *SIGCSE Bulletin, 31*(3), 155-158. doi:10.1145/384267.305904

Dancey, C. P., & Reidy, J. (2011). *Statistics Without Maths for Psychology* (fifth ed.). London, UK: Prentice Hall.

De Leeuw, E. D., Hox, J., & de Leeuw, E. D. (2012). *International Handbook of Survey Methodology.* New York. doi:10.4324/9780203843123

Deitel, P., & Deitel, H. (2011). *Java: How to Program* (9th ed.). Upper Saddle River, USA: Prentice Hall Press.

Delhij, A., van Solingen, R., & Wijnands, W. (2015). *The eduScrum Guide - "The rules of the Game"*. Obtido de http://eduscrum.nl/en/file/CKFiles/The_eduScrum_Guide_EN_1.2.pdf

Deo, R. C. (2015). Machine Learning in Medicine. *Circulation, 132*(20), 1920-1930.

Descartes, R. (2008). *Discurso do Método (1637).* Edições 70.

Diakopoulos, N., Cass, S., & Romero, J. (2014). Data-Driven Rankings: The Design and Development of the IEEE Top Programming Languages News App. *Proceedings Symposium on Computation + Journalism.*

Díaz de Rada, V., & Domínguez-Álvarez, J. A. (2014). Response quality of self-administered questionnaires: A comparison between paper and web questionnaires. *Social Science Computer Review, 32*(2).

Diwakar, S., Parasuram, H., Medini, C., Raman, R., Nedungadi, P., Wiertelak, E., . . . Nair, B. (2014). Complementing Neurophysiology Education for Developing Countries via Cost-Effective Virtual Labs: Case Studies and Classroom Scenarios. *Journal of undergraduate neuroscience education, 12*(2), A130–A139.

Dorine, A., Blair, N., & Preece, J. (2003). Conducting Research on the Internet: Online Survey Design, Development and Implementation Guideline. *International Journal of Human-Computer Interaction*.

Dougiamas, M., & Taylor, P. C. (2003). Moodle: Using Learning Communities to Create an Open Source Course Management System. Em D. Lassner, & C. McNaught (Ed.), *Proceedings of EDMEDIA 2003 - World Conference on Educational Multimedia, Hypermedia and Telecommunications,* (pp. 171-178). Honolulu, Hawaii.

Duminuco, V. (2000). *The Jesuit Ratio Studiorum: 400th Anniversary Perspectives.* New York: Fordham University Press.

Dziuban, C. D., & Shirkey, E. C. (1974). When is a correlation matrix appropriate for factor analysis? Some decision rules. *Psychological Bulletin, 81*(6), 358–361. doi:doi.org/10.1037/h0036316

Edwards, S. H., & Pérez-Quiñones, M. A. (2008). Web-CAT: automatically grading programming assignments. *SIGCSE Bulletin, 40*(3), 328-328. doi:10.1145/1597849.1384371

Engelhardt, A., & Balanskat, K. (2015). *Computing our future. Computer programming and coding Priorities, school curricula and initiatives across Europe.* European Schoolnet.

Esclapez, A. C. (2008). *La enseñanza que no se ve. Educación informal en el siglo XXI* (1ª ed.). Madrid: Narcea S. A. de Ediciones.

Esengün, M., & Ince, G. (2018). The Role of Augmented Reality in the Age of Industry 4.0. Em A. Ustundag, & E. Cevikcan, *Industry 4.0: Managing The Digital Transformation.* Springer. doi:10.1007/978-3-319-57870-5_12

Eshach, H. (April de 2007). Bridging In-school and Out-of-school Learning: Formal, Non-Formal, and Informal Education. *Journal of Science Education and Technology, 16*(2), 171–190. doi:10.1007/s10956-006-9027-1

Farshida, M., Paschena, J., Erikssonb, T., & Kietzmann, J. (2018). Goboldly! Explore augmented reality(AR), virtualreality(VR), and mixed reality(MR) for business. *Business Horizons*, *61*(5), pp. 657-663.

Ferreira, E. P., & Martins, Â. (2016). EduScrum – The Empowerment of Students in Engineering Education? *Proceedings of the 12th International CDIO Conference, Turku University of Applied Sciences.* Turku, Finland.

Ferreira, M. J., & Campos, P. (2009). Inquérito Estatístico: uma introdução à elaboração de questionários, amostragem, organização e apresentação dos resultados. *Um mundo para conhecer os números*, p. 214.

Fiolhais, C. (2005). Informática, sociedade e educação científica. *XVI Simpósio Nacional de Ensino da Física.* Rio de Janeiro, Brasil. Obtido de http://hdl.handle.net/10316/12376

Flanagan, D., & Matsumoto, Y. (2008). *The Ruby Programming Language: Everything You Need to Know*. O'Reilly.

Fonte, D., da Cruz, D., Gançarski, A. L., & Henriques, P. R. (2013). A Flexible Dynamic System for Automatic Grading of Programming Exercises. Em J. P. Leal, R. Rocha , & A. Simões (Ed.), *2nd Symposium on Languages, Applications and Technologies. 29*, pp. 129-144. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.SLATE.2013.129

França, A. B., & Soares, J. M. (2012). Sistema de apoio a atividades de laboratório de programação com suporte ao balanceamento de carga e controle de plágio. *Anais do 23º Simpósio Brasileiro de Informática na Educação (SBIE 2012).*

Fuentes-Rosado, J. I., & Moo-Medina, M. (2017). Dificultades de aprender a programar. *Revista Educación En Ingeniería*, *12*(24). doi:10.26507/rei.v12n24.728

Funabiki, N., Matsushima, Y., Nakanishi, T., Watanabe, K., & Amano, N. (2013). A Java Programming Learning Assistant System Using Test-Driven Development Method. *IAENG International Journal of Computer Science, 40*(1), 38-46.

Gabbrielli, M., & Martini, S. (2010). *Programming Languages: Principles and Paradigms.* Springer-Verlag.

Gilb, T., & Susannah, F. (1988). *Principles of Software Engineering Management.* Addison-Wesley.

Gohn, M. (2006). Educação não formal, participação da sociedade civil e estruturas colegiadas nas escolas. *Ensaio: Avaliação e Políticas Públicas em Educação, 14*(50), pp. 27-38. Obtido de http://www.scielo.br/pdf/ensaio/v14n50/30405.pdf

Gomes, A. (1967). A telescola em Portugal. (M. d. Nacional, Ed.) *Boletim do Instituto de Orientação Profissional*, p. 143.

Gomes, A. (2010). Dificuldades de aprendizagem de programação de computadores: contributos para a sua compreensão e resolução. *Tese de Doutoramento*. Faculdade de Ciências e Tecnologia, Universidade de Coimbra.

Gomes, A., & Mendes, A. J. (2007). Learning to program–difficulties and solutions. *Proceedings of International Conference on Engineering Education-ICEE '07*, (pp. 283-287). Coimbra.

Gouveia, L. B. (2006). *Negócio Electrónico - conceitos e perspectivas de desenvolvimento.* SPI - Sociedade Portuguesa de Inovação.

Graham, B. B. (2004). *Detail Process Charting: Speaking the Language of Process.* John Wiley & Sons, Inc.

Gravier, C., Fayolle, J., Ates, M., & Lardon, J. (2008). State of the art about remote laboratories paradigms - foundations of ongoing mutations. *International Journal of Online Engineering (iJOE), 4*(1), 19-25.

Gupta, S., & Gupta, A. (2018). E-Assessment Tools for Programming Languages: A Review. *Proceedings of the First International Conference on Information Technology and Knowledge Management*, *14*, pp. 65-70.

Gürer, D. W. (1995). Pioneering Women in Computer Science. *Communications of the ACM, 38*(1), 45-54. doi:10.1145/204865.204875

Guzmán, J. G., Ramos, J. S., Seco, A. A., & Esteban, A. S. (2011). Success factors for the management of global virtual teams for software development. *International Journal of Human Capital and Information Technology Professionals, 2*(2), 48-59.

Heflin, H., Shewmaker, J., & Nguyen, J. (2017). Impact of mobile technology on student attitudes, engagement, and learning. *Computers & Education, 107*, pp. 91-99. doi:10.1016/j.compedu.2017.01.006

Heng, P., Joy, M., Boyat, R., & Griffiths, N. (2005). *Evaluation of the BOSS Online Submission and Assessment System.* Computer Science Research Report, University of Warwick, Department of Computer Science.

Heradio, R., de la Torre, L., & Dormido, S. (2016). Virtual and remote labs in control education: A survey. *Annual Reviews in Control, 42*, 1-10. doi:10.1016/j.arcontrol.2016.08.001.

Higgins, C., Hegazy, T., Symeonidis, P., & Tsintsifas, A. (01 de 09 de 2003). The CourseMarker CBA System: Improvements over Ceilidh. *Education and Information Technologies, 8*(3), 287-304. doi:10.1023/A:1026364126982

Highsmith, J. A. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* New York, USA: Dorset House Publishing Co., Inc.

Highsmith, J. A. (2002). *Agile Software Development Ecosystems.* Boston, Massachusetts, USA: Addison Wesley.

Hill, M. M., & Hill, A. (2008). *Investigação por Questionário(2ª Edição).* Edições Silabo.

Hollingsworth, J. (1960). Automatic graders for programming classes. *Communications of the ACM*, pp. 528-529.

Hongyu, K. (2018). Análise Fatorial Exploratória: resumo teórico, aplicação e interpretação. *E&S Engineering and Science, 4*(7).

Horstmann, C. S. (2013). *Big Java: Late Objects.* John Wiley & Sons.

Huang, Y.-M., Chiu, P.-S., Liu, T.-C., & Chen, T.-S. (2011). The design and implementation of a meaningful learning-based evaluation method for ubiquitous learning . *Computers & Education, 57*(4), 2291--2302. doi:10.1016/j.compedu.2011.05.023

Hudson, P. (2005). *PHP in a Nutshell.* Sebastopol, California, USA: O'Reilly Media, Inc.

Hutcheson, G. D., & Sofroniou, N. (1999). *The multivariate social scientist. introductory statistics using generalized linear models.* London: Sage Publications.

Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of Recent Systems for Automatic Assessment of Programming Assignments. *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)* (pp. 86-93). ACM. doi:10.1145/1930464.1930480

ISO-International Organization for Standardization. (2012). ISO/IEC 30170:2012. *Information Technology - Programming languages - Ruby*.

Jackson, D., & Usher, M. (1997). Grading Student Programs using ASSYST. Em J. E. Miller (Ed.), *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '97)* (pp. 335-339). San Jose, California, USA: ACM. doi:10.1145/268084.268210

Jackson, S. (2016). *The Art of Neural Networks Using C Sharp.* Createspace Independent Pub.

Jardim, S. V. (2013). The electronic health record and its contribution to healthcare information systems interoperability. *Procedia Technology, 9*, 940-948. doi:10.1016/j.protcy.2013.12.105

Jazdi, N. (2014). Cyber Physical Systems in the Context of Industry 4.0. *IEEE International Conference on Automation, Quality and Testing, Robotics.* Cluj-Napoca, Romania.: IEEE.

Johnson, S. C., & Kernighan, B. W. (1973). *The programming language B.* Technical Report, AT&T Bell Laboratories.

Joy, M., & Luck, M. (1995). On-line Submission and Testing of Programming Assignments. (J. Hart, Ed.) *Innovations in the Teaching of Computing, 1*(88), pp. 95-103.

Joy, M., & Luck, M. (1999). Plagiarism in programming assignments. *IEEE Transactions on Education, 42*(2), pp. 129-133. doi:10.1109/13.762946

Joy, M., Griffiths, N., & Boyatt, R. (2005). The Boss Online Submission and Assessment System. *Journal on Educational Resources in Computing, 5*(3). doi:10.1145/1163405.1163407

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior, 52*(200-210). doi:doi.org/10.1016/j.chb.2015.05.047

Kelleher, J. (2014). Employing Git in the Classroom. *World Congress on Computer Applications and Information Systems (WCCAIS)*, (pp. 1-4). doi:10.1109/WCCAIS.2014.6916568

Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (second ed.). Englewood Cliffs, New Jersey, USA: Prentice-Hall.

Kistermann, F. W. (2005). Hollerith punched card system development (1905-1913). *IEEE Annals of the History of Computing, 27*(1), pp. 56-66.

Kline, P. (2000). *The Handbook of Psychological Testing* (second ed.). Londres, UK: Routledge.

Knuth, D. E. (1997). *The Art of Computer Progmmming, Volume 1: Fundamental Algorithms* (Third edition ed.). Addison-Wesley.

Koulouri, T., Lauria, S., & Macredie, R. D. (2015, February). Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. *ACM Transactions on Computing Education (TOCE), v.14 n.4*, pp. 1-28. doi:10.1145/2662412

Kowalski, R., & Kuehner, D. (1971). Linear Resolution with Selection Function. *Artificial Intelligence 2*, pp. 227–260. doi:10.1016/0004-3702(71)90012-9

Krause, P. K., Larisch, L., & Salfelder, F. (2019). The tree-width of C. *Discrete Applied Mathematics*. doi:10.1016/j.dam.2019.01.027

Krochmalski, J. (2014). *IntelliJ IDEA Essentials.* UK: Packt Publishing.

Kuhn, T. S. (1962). *The Structure of Scientific Revolutions.* Chicago, USA: University of Chicago Press.

Kumar, K., & Dahiya, S. (2017). Programming Languages: A Survey. *International Journal on Recent and Innovation Trends in Computing and Communication , 5*(5), 307-313.

Kurtz, T. E. (1978). BASIC. (ACM, Ed.) *SIGPLAN Notices, 13*(8), 103-118. doi:10.1145/960118.808376

Kuusinen, K., Gregory, P., Sharp, H., & Barroca, L. (2017). Knowledge Sharing in a Large Agile Organisation: A Survey Study. (H. Baumeister, H. Lichter, & M. Riebisch, Edits.) *Agile Processes in Software Engineering and Extreme Programming*, pp. 135-150. doi:10.1007/978-3-319-57633-6 9

Lagarto, J. R. (2002). *Ensino à distância e formação contínua: uma análise prospectiva sobre a utilização do ensino a distância na formação profissional contínua de activos em Portugal.* Lisboa: Instituto para a Inovação na Formação.

Lagarto, J., & Andrade, A. V. (2009). Sistemas de Gestão de Aprendizagem em Elearning. Em G. L. Miranda (Ed.), *Ensino Online e Aprendizagem Multimédia* (pp. 56-80). Relógio D´Água Editores.

Lancaster, T., & Culwin, F. (2004). A Comparison of Source Code Plagiarism Detection Engines. *Computer Science Education, 14*(2), pp. 101-112. doi:10.1080/08993400412331363843

Larman, C. (2003). *Agile and Iterative Development: A Manager's Guide.* (A. Cockburn, & J. Highsmith, Edits.) Boston, USA: Addison-Wesley.

Larman, C., & Basili, V. (2003). Iterative and Incremental Development: A Brief History. *Computer, 36*(6), pp. 47-56.

Leal, J. P., & Santos, A. M. (2008). Avaliação de Satisfação dos Utilizadores do Mooshak. *Proceedings of CIAWI - Conferência Ibero-Americana WWW/Internet 2008.* Lisboa, Portugal: IADIS.

Leal, J. P., & Silva, F. (2003). Mooshak: a Web-based multi-site programming contest system. *Software: Practice and Experience, 33*(6), 567-581. Obtido de https://doi.org/10.1002/spe.522

Leal, J. P., & Silva, F. (2008). Using Mooshak as a competitive learning tool. *Proceedings of ACM-ICPC Competitive Learning Institute Symposium (CLIS 2008).* Banff, Canada.

Leite, V. M., Senefonte, H. d., Barbosa, C., & Seabra, R. D. (2013). VisuAlg: Estudo de Caso e Análise de Compilador destinado ao ensino de Programação. *Nuevas Ideas en Informática Educativa*, pp. 637-640.

Lerdorf, R. (2000). *PHP : pocket reference* (first ed.). Sebastopol, California, USA: O'Reilly Media, Inc.

Lesh, R. A., & Zawojewski, J. S. (2007). *The Handbook of Research on Mathematics Teaching and Learning* (second ed.). (F. K. Lester, Ed.) Charlotte, North Carolina, USA: Information Age Publishing.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology, 22*(140), pp. 44-53.

Lima, J. R., & Capitão, Z. A. (2003). *E-learning e E-conteúdos : aplicações das teorias tradicionais e modernas de ensino e aprendizagem à organização e estruturação de e-cursos.* Lisboa: Centro Atlântico.

Lindsey, C. H. (1996). A History of ALGOL 68. Em T. J. Bergin Jr., & R. G. Gibson Jr, *History of Programming languages---II* (pp. 27-96). New York, USA: ACM. doi:10.1145/234286.1057810

Littlejohn, A., & Pegler, C. (2007). *Preparing for Blended e-Learning.* New York: Routledge.

Lopes, A. P. (2011). *Proceedings of INTED2011 – International Technology, Education and Development Conference*, (pp. 970-976). Valencia, Spain.

Lovrenčić, A., Konecki, M., & Orehovački, T. (2009). 1957-2007: 50 Years of Higher Order Programming Languages. *Journal of Information and Organizational Sciences, 33*(1). Obtido de https://hrcak.srce.hr/38734

Lustig, T. A. (2012). *The role of telehealth in an evolving health care environment: workshop summary.* Washington (DC), USA: The National Academies Press.

Lutz, M. (2013). *Learning Python* (5th ed.). Sebastopol, USA: O'Reilly Media, Inc.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., . . . Szabo, C. (2018). Introductory Programming: A Systematic Literature Review. *Proceedings of the 2018 ITiCSE Conference on Working Group Reports* (pp. 55-106). New York, USA: ACM. doi:10.1145/3293881.3295779

Lykke, M., Coto, M., Jantzen, C., Mora, S., & Vandel, N. (2015). Motivating Students through Positive Learning Experiences: A Comparison of Three Learning Designs for Computer Programming Courses. *Journal of Problem Based Learning in Higher Education, 3*(2), 80–108.

Maciel, D. L., Soares, J. M., França, A. B., & Gomes, D. G. (2012). Análise de Similaridade de códigos-fonte como estratégia para o acompanhamento de atividades de laboratório de programação. *RENOTE - Revista Novas Tecnologias na Educação*. doi:10.22456/1679-1916.36453

Manev, K. N., & Maneva, N. (2014). On Applying Design Pattern Approach to Reengineering COBOL Programs. *Proceedings of the International conference on Computing Technology and Information Management*, (pp. 125-134). Dubai, UAE.

Manso, A., Marques, C. G., & Dias, P. (2010). Portugol IDE v3.x: A new environment to teach and learn computer programming. *IEEE EDUCON Engineering Education 2010 Proceedings* (pp. 1007–1010). IEEE. doi:10.1109/EDUCON.2010.5492469

Maroco, J., & Garcia-Marques, T. (2006). Qual a fiabilidade do alfa de Cronbach? Questões antigas e soluções modernas? (I. S. Aplicada, Ed.) *Laboratório de Psicologia, 4*(1), pp. 65-90.

Marques, C. C. (2011). Desenvolvimento e implementação de um modelo de bended-learning com objectos de aprendizagem no ensino superior . *Tese de Doutoramento*.

Martin, F. (2008). Blackboard as the learning management system of a computer literacy course. *MERLOT Journal of Online Learning and Teaching, 4*(2), 138-145.

Martin, F., Wang, C., Petty, T., Wang, W., & Wilkins, P. (2018). Middle School Students' Social Media Use. *Journal of Educational Technology & Society, 21*(1), 213-224.

Maslow, A. H. (1987). *Motivation and personality* (third ed.). New York, USA: Addison-Wesley.

McKnight, K., O'Malley, K., Ruzic, R., Horsley, M. K., Franey , J. J., & Bassett, K. (2016). Teaching in a Digital Age: How Educators Use Technology to Improve Student Learning. (Routledge, Ed.) *Journal of Research on Technology in Education, 48*(3), 194-211. doi:10.1080/15391523.2016.1175856

Mens, T., Fernandez-Ramil, J., & Degrandsart, S. (2008). The evolution of Eclipse. *Proceedings of the International Conference on Software Maintenance* (pp. 386–395). IEEE.

Mesquita, A., Moreira, F., & Peres, P. (2016). Customized learning environment: A new approach. *Information Systems and Technologies (CISTI), 2016 11th Iberian Conference, 1*, pp. 228-231. doi:10.1109/CISTI.2016.7521621

Meyerovich, L. A., & Rabkin, A. S. (2013). Empirical analysis of programming language adoption. *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications (OOPSLA '13)*, pp. 1-18. doi:10.1145/2509136.2509515

Miranda, G. L. (2009). Concepção de conteúdos e cursos online. Em G. L. Miranda, *Ensino online e aprendizagem multimédia* (pp. 81-110). Lisboa: Relógio D' Água.

Mishra, A., & Dubey, D. (2013). A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios. *International Journal of Advance Research in Computer Science and Management Studies*, *1*(5), 64-69.

Moniruzzaman, A. B., & Hossain, S. A. (2013). Comparative study on agile software development methodologies. *Global Journal of Computer Science and Technology, 13*(7).

Monteiro, A., Moreira, J. A., & Lencastre, J. A. (2015). *Blended (E) Learning na Sociedade Digital.* Santo Tirso: Whitebooks.

Moodle. (2019). Obtido em 16 de 02 de 2019, de Moodle: https://moodle.org/plugins/

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics, 38*, pp. 114-117.

Moore, M. G., & Kearsley, G. (2011). *Distance Education: A Systems View of Online Learning* (Third ed.). Cengage Learning.

Moore, M. G., & Thompson, M. M. (1997). *The Effects of Distance Learning.* Pennsylvania State University: American Center for the Study of Distance Education.

Morris, D. S. (2003). Automatic grading of student's programming assignments: an interactive process and suite of programs. *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference. 3*, pp. S3F.1 - S3F.5. Boulder, Colorado, USA: IEEE Computer Society. doi:10.1109/FIE.2003.1265998

Moström, J. E. (2011). A Study of Student Problems in Learning to Program. *PhD dissertation*. Department of Computing Science, Umeå University, Umeå, Sweden.

Muir-Herzig, R. G. (2004). Technology and its impact in the classroom. *Computers & Education, 42*(2), 111–131.

Nazário, D. C., & Souza, A. (2010). BOCA-LAB: Corretor automático de Código adaptado ao Ensino de Linguagem de Programação. *Anais de Computer on the Beach 2010* . Florianópolis, Santa Catarina, Brasil.

Nemat, R. (2011). Taking a look at different types of e-commerce. *World Applied Programming, 1*(2), 100-104. doi:10.1.1.684.6401

Norrish, M. (1998). *C formalised in HOL.* University of Cambridge.

Noschang, L. F., Pelz, F., Jesus, E. A., & Raabe, A. L. (2014). Portugol Studio: Uma IDE para iniciantes em programação. *XXII Workshop sobre Educação em Computação,* (pp. 1287–1296).

Nunnally, J. C. (1994). *Psychometric theory.* New York: McGraw-Hill.

O'Hearn, P. W., & Tennent, R. D. (1997). *Algol-like Languages.* Birkhäuser. doi:10.1007/978-1-4612-4118-8

O'Regan, G. (2013). John Backus. Em *Giants of Computing* (pp. 31-34). Springer. doi:doi.org/10.1007/978-1-4471-5340-5_7

Ochse, R. A. (1990). *Before the gates of excellence: the determinants of creative genius.* Cambridge, UK: Cambridge University Press.

OECD. (2013). *PISA 2012 Assessment and Analytical Framework: Mathematics, Reading, Science, Problem Solving and Financial Literacy.* Paris, França: OECD Publishing. doi:10.1787/9789264190511-en

Oracle. (2016). *NetBeans Developing Applications with NetBeans IDE 8.2.* Obtido de https://docs.oracle.com/netbeans/nb82/netbeans/NBDAG/NBDAG.pdf

Paiva, J. P. (2015). *Redes de Energia Eléctrica - Uma análise sistémica* (4ª ed.). Lisboa, Portugal: IST - Instituto Superior Técnico.

Palacios, R. C., Lumbreras, C. C., Acosta, P. S., García-Peñalvo, F. J., & Tovar, E. (2014). Project managers in global software development teams: a study of the effects on productivity and performance. *Software Quality Journal, 22*(1), 3-19. doi:10.1007/s11219-012-9191-x

Palmer, S. R., & Felsing, J. M. (2002). *A Practical Guide to Feature-Driven Development.* Prentice Hall.

Palmquist, M. S., Lapham, M. A., Miller, S., Chick, T., & Ozkaya, I. (2013). *Parallel Worlds : Agile and Waterfall Differences and Similarities.* Carnegie Mellon University, Software Engineering Institute. Obtido de https://apps.dtic.mil/dtic/tr/fulltext/u2/a610501.pdf

Papaspyrou, N. S. (1998). A Formal Semantics for the C Programming Language. *Doctoral Dissertation*. National Technical University of Athens.

Parker, K. R., & Davey, B. (2012). The History of Computer Language Selection. *Reflections on the History of Computing*, pp. 166-179. doi:10.1007/978-3-642-33899-1_12

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., . . . Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin, 39*(4), pp. 204-223. doi:10.1145/1345443.1345441

Pelz, F. D., Jesus, E. A., & Raabe, A. L. (2012). Um Mecanismo para Correção Automática de Exercícios Práticos de Programação Introdutória. *Simpósio Brasileiro de Informática na Educação - SBIE).* Obtido de https://www.br-ie.org/pub/index.php/sbie/article/view/1780/1541

Peres, P., & Pimenta, P. (2011). *Teorias e práticas de b-learning.* Lisboa: Edições Sílabo.

Peres, P., Mesquita, A., & Pimenta, P. (2015). *Guia Prático de e-Learning: Casos práticos nas organizações.* Porto: Vida Económica.

Perlis, A. J. (1995). Talk on Computing in the fiftie. *Computer Pioneers*, pp. 545-556.

Perraton, H. (2007). *Open and Distance Learning in the Developing World* (2nd ed.). Routledge.

Pettit, R., & Prather, J. (2017). Automated assessment tools: too many cooks, not enough collaboration. *Journal of Computing Sciences in Colleges, 32*(4), 113-121.

Piteira, M., & Costa, C. (2013). Learning Computer Programming: Study of Difficulties in Learning Programming. *Proceedings of the 2013 International Conference on Information Systems and Design of Communication (ISDOC '13)* (pp. 75-80). Lisboa, Portugal: ACM. doi:10.1145/2503859.2503871

Piteira, M., & Costa, C. J. (2017). Gamificação: Framework Concetual para Cursos Online de Aprendizagem da Programação. *12th Iberian Conference on Information Systems and Technologies (CISTI).* Lisboa, Portugal: IEEE. Obtido de http://hdl.handle.net/10400.26/18601

Plazzotta, F., Luna, D., & Quirós, F. G. (2015). Sistemas de información en salud: integrando datos clínicos en diferentes escenarios y usuarios. *Revista Peruana de Medicina Experimental y Salud Pública, 32*(2), pp. 343-351. doi:10.17843/rpmesp.2015.322.1630

Pohuba, D., Dulík, T., & Janků, P. (2014). Automatic evaluation of correctness and originality of source codes. *10th European Workshop on Microelectronics Education (EWME 2014),* (pp. 49-52).

Poppendieck, M., & Cusumano, M. A. (2012). Lean Software Development: A Tutorial. *IEEE Software, 29*(5), 26-32. doi:10.1109/MS.2012.107

Porter, M. E., & Heppelmann, J. E. (2015). How Smart, Connected Products Are Transforming Companies. *Harvard Business Review , 93*(10), pp. 96-114.

Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach* (Seventh Edition ed.). New York, USA: McGraw Hill.

Qin, J., Liu, Y., & Grosvenor, R. (2016). A Categorical Framework of Manufacturing for Industry 4.0 and Beyond. *Sixth International Conference on Changeable, Agile, Reconfigurable and Virtual Production. 52*, pp. 173-178. Elsevier Science Bv.

Queirós, R. A. (2012). A framework for practice-based learning applied to computer programming. *Tese de Doutoramento*. Faculdade de Ciências da Universidade do Porto.

Queirós, R. A., & Leal, J. P. (2012). PETCHA: A Programming Exercises Teaching Assistant. *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (pp. 192-197). Haifa, Israel: ACM. doi:10.1145/2325296.2325344

Queirós, R., & Leal, J. P. (2013). crimsonHex: A learning objects repository for programming exercises. *Software: Practice and Experience, 43*(8). doi:10.1002/spe.2132

Rabai, L. B., Cohen, B., & Mili, A. (2015). Programming Language Use in US Academia and Industry. *Academia and Industry, Informatics in Education*, 143–160. doi:10.15388/infedu.2015.09

Ralston, A., & Meek, C. L. (1976). *Encyclopedia of Computer Science.* New York, USA: Petrocelli/Charter.

Rashed, M. G. (2012). Python in Computational Science: Applications and Possibilities. *International Journal of Computer Applications, 46*(20), 26-30.

Regueras, L. M., de Castro, J. P., Verdú, E., Perez, M. Á., & Verdú, M. J. (2009). A Proposal of User Interface for a Distributed Asynchronous Remote Evaluation System: An Evolution of the QUESTOURnament Tool. *Proceedings of the 9th IEEE international conference on advanced learning technologies (ICALT 2009)*, (pp. 75-77).

Revilla, M. A., Manzoor, S., & Liu, R. (2008). Competitive Learning in Informatics: The UVa Online Judge Experience. *Olympiads in Informatics, 2*, pp. 131–148.

Richard, L., & Harlow, J. (2016). Coderunner: a tool for assessing computer programming skills. *ACM Inroads, 7*(1), 47-51. doi:10.1145/2810041

Richards, M., & Whitbey-Strevens, C. (1979). *BCPL: The Language and its Compiler.* Cambridge, UK: Cambridge University Press.

Ritchie, D. M. (1993). The Development of the C Language. (ACM, Ed.) *ACM SIGPLAN Notices, 28*(3), pp. 201-208.

Rodrigues, S., Rocha, Á., & Abreu, A. (2017). The use of moodle in higher education evolution of teacher's practices over time. *Information Systems and Technologies (CISTI), 2017 12th Iberian Conference* (pp. 1-4). Lisboa, Portugal: IEEE.

Rodríguez-del-Pino, J. C., Rubio-Royo, E., & Hernández-Figueroa, Z. J. (2010). VPL: Laboratorio Virtual de Programación para Moodle. *Actas de Jornadas de Enseñanza Universitaria de la Informática.*

Rodríguez-del-Pino, J. C., Rubio-Royo, E., & Hernández-Figueroa, Z. J. (2011). Uses of VPL. *5th International Technology, Education and Development Conference (INTED)*, (pp. 743-748). Valencia, Espanha.

Rodríguez-del-Pino, J. C., Rubio-Royo, E., & Hernández-Figueroa, Z. J. (2012). A virtual programming lab for moodle with automatic assessment and anti-plagiarism features. *Proceedings of the 2012 International Conference on e-Learning, e-Business, Enterprise Information System.* Las Vegas. Obtido de https://vpl.dis.ulpgc.es/

Rojas, R., & Hashagen, U. (2000). *The first computers: history and architectures.* Cambridge, Massachusetts, United States of America: MIT Press.

Rosenberg, M. J. (2006). *eBeyond E-Learning: Approaches and Technologies to Enhance Organizational Knowledge, Learning, and Performance.* San Francisco: Jonh Wiley & Sons, Inc.

Royce, W. W. (1970). Managing the development of large software systems: Concepts and techniques. *Proceedings of Western Electronic Show and Convention (WesCon).* IEEE Computer Society Press.

Rubin, K. S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process.* Boston: Addison-Wesley Professional.

Saccol, A., Schlemmer, E., & Barbosa, J. (2011). *M-learning e u-learning: novas perspectivas da aprendizagem móvel e ubíqua.* São Paulo, Brasil: Pearson. Prentice Hall.

Sammet, J. E. (1972). Programming Languages: History and Future. *Communications of the ACM, 15*(7), pp. 601-610. doi:10.1145/361454.361485

Sammet, J. E. (1978a). Roster of Programming Languages for 1976-77. *ACM SIGPLAN Notices, 13*(11), pp. 56-85.

Sammet, J. E. (1978b). The Early History of COBOL. Em R. L. Wexelblat, *History of Programming Languages* (pp. 199-243). New York, USA: ACM. doi:10.1145/800025.1198367

Santucci, U. (2010). Problem setting. Obtido em 10 de 12 de 2018, de http://www.giovannifazzone.eu/problem_setting.pdf

Satav, S. K., Satpathy, S. K., & Satao, K. (2011). A Comparative Study and Critical Analysis of Various Integrated Development Environments of C, C++, and Java Languages for Optimum Development. *IT-Contemporary & Future Technologies for Social Change*, pp. 9-15.

Schleicher, A. (2019). Children, technology and teaching. Em A. Schleicher, *Helping our Youngest to Learn and Grow: Policies for Early Learning.* Paris, France. doi:10.1787/f21353a9-en

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*(61), pp. 85-117.

Schorsch, T. (1995). CAP: an automated self-assessment tool to check Pascal programs for syntax, logic and style errors. (C. M. White, J. E. Miller, & J. Gersting, Edits.) *Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education (SIGCSE '95)*, pp. 168-172.

Sebesta, R. W. (2016). *Concepts of Programming Languages* (11th Edition ed.). Pearson.

Sellink, A., & Verhoef, C. (1997). *Reflections on the evolution of COBOL.* University of Amsterdam.

Setzer, V. W. (2001). *Meios eletrônicos e educação: uma visão alternativa* (Escrituras ed., Vol. 10). São Paulo, Brasil.

Shapiro, J. R. (2002). *Visual Basic .net The complete Reference.* USA: McGraw-Hill.

Sharan, K. (2017). *Beginning Java 9 Fundamentals: Arrays, Objects, Modules, JShell, and Regular Expressions* (second ed.). New York, USA: Apress. doi:10.1007/978-1-4842-2902-6

Sherman, M., Bassil, S., Lipman, D., Tuck, N., & Martin, F. (2013). Impact of auto-grading on an introductory computing course. *Journal of Computing Sciences in Colleges, 28*(6), 69-75.

Shih, Y. E., & Mills, D. (2007). Setting the new standard with mobile computing in online learning. *International Review of Research in Open and Distance Learning, 8*(2), pp. 1-6.

Siegfried, R. M., Greco, D., Miceli, N., & Siegfried, J. (2012). Whatever Happened to Richard Reid's List of First Programming Languages? *Information Systems Education Journal, 10*(4). Obtido de Information Systems Education Journal

Silva, C. A., Santos, E. L., Angelo, L. M., Oliveira, M. A., & Moraes, R. V. (2016). A utilização do SCRUM como recurso educacional no processo de aprendizagem em Engenharia de Software. *International Journal of Alive Engineering Education, 1*(2), 87-102.

Singh, D. (2017). An Empirical Study of Programming Languages from the Point of View of Scientific Computing. *IJISET - International Journal of Innovative Science, Engineering & Technology, 4*(6), 367–371.

Skiena, S. S. (2008). *The Algorithm Design Manual* (Second ed.). London, UK: Springer-Verlag.

Skiena, S. S., & Revilla, M. A. (2003). *Programming Challenges: The Programming Contest Training Manual*. (D. Gries, & F. B. Schneider, Edits.) New York, USA: Springer. doi:10.1145/945526.945539

Skinner, B. F. (24 de outrubro de 1958). Science New Series. *Teaching machines, 128*(3330), pp. 969-977.

Sklar, D., & Trachtenberg, A. (2014). *PHP Cookbook - Solutions & Examples for PHP* (Third ed.). O'Reilly Media, Inc.

Skūpas, B., Čaplinskas, A., Augutis, J., Bareiša, E., Kulvietis, G., Marcinkevičius, V., . . . Šeinauskas, R. (2013). A Method for Semi-Automatic Evaluation and Testing of Programming Assignments. *Summary of Doctoral Dissertation*. Vilnius, Lithuania.

Souza, C. M. (setembro de 2009). VisuAlg - Ferramenta de Apoio ao Ensino de Programação. *Revista TECCEN , 2*(2).

Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., & Padua-Perez, N. (2006). Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses. *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITICSE '06)*, pp. 13-17. doi:10.1145/1140124.1140131

Srikant, S., & Aggarwal, V. (2014). A system to grade computer programming skills using machine learning. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1887-1896). ACM. doi:10.1145/2623330.2623377

Striewe, M., Balz, M., & Goedicke, M. (2009). A Flexible and Modular Software Architecture for Computer Aided Assessments and Automated Marking. *Proceedings of the First International Conference on Computer Supported Education (CSEDU). 2*, pp. 54-61. Lisboa, Portugal: INSTICC. Obtido de https://www.s3.uni-duisburg-essen.de/en/jack/index.html

Stroustrup, B. (1993). A History of C++: 1979-1991. *ACM SIGPLAN NOTICES, 28*, 271-297.

Stroustrup, B. (2013). *The C++ Programming Language* (fourth ed.). Addison-Wesley.

Summerfield, M. (2007). *Rapid GUI Programming with Python and Qt*. New Jersey, USA: Prentice Hall.

Sung, Y.-T., Chang, K.-E., & Liu, T.-C. (2016). The effects of integrating mobile devices with teaching and learning on students' learning performance: A meta-analysis and research synthesis. *Computers & Education*(94), pp. 252-275. doi:10.1016/j.compedu.2015.11.008

Sutherland, J. (2014). *Scrum: The Art of Doing Twice the Work in Half the Time*. New York, USA: Crown Business.

Sutherland, J., & Schwaber, K. (2007). *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process*. Obtido de Scrum Inc.: https://klevas.mif.vu.lt/~adamonis/pkp/1415p/lit/ScrumPapers.pdf

Tabachnick, B. G., & Fidell, L. S. (2006). *Using Multivariate Statistics* (Fifth ed.). Pearson.

Tavares, P. C. (04 de 2018). O Impacto da Animação e da Avaliação Automática na Motivação para o Ensino da Programação. *Tese de Doutoramento*. Universidade do Minho.

Tavares, P. C., Henriques, P. R., & Gomes, E. F. (2015). Animation and automatic evaluation in supporting the teaching of programming. *10th Iberian Conference on Information Systems and Technologies, CISTI 2015*. Lisboa. doi:10.1109/CISTI.2015.71705

Thayer-Hart, N., Dykema, J., Elver, K., Schaeffer, N. C., & John, S. (2010). *Survey Fundamentals - A guide to designing and implementing surveys, Office of Quality Improvement.* University of Wisconsin Survey Center.

Thiébaut, D. (2015). Automatic evaluation of computer programs using Moodle's virtual programming lab (VPL) plug-in. *Journal of Computing Sciences in Colleges, 30(6),* 145-151.

Tiantian, W., Xiaohong, S., Peijun, M., Yuying, W., & Kuanquan, W. (2009). AutoLEP: An Automated Learning and Examination System for Programming and its Application in Programming Course. *Proceedings of the 2009 First International Workshop on Education Technology and Computer Science. 1*, pp. 43-46. IEEE. doi:10.1109/ETCS.2009.18

Tiobe - The Software Quality Company. (2018). *TIOBE Index*. Retrieved 05 06, 2018, from https://www.tiobe.com/tiobe-index/

Trilla-Bernet, J., Salvat, B. G., Palma, F. L., García, M. M., aaasa , f., & fdg gfgfg, g. (2014). *Educacion Fuera de la Escuela. Ámbitos no formales y educación social.* Ariel Educación.

Trindade, A. R. (1990). *Introdução à comunicação educacional.* Universidade Aberta.

Truong, N., Bancroft, P., & Roe, P. (2003). A Web Based Environment for Learning to Program. Em M. J. Oudshoorn (Ed.), *Proceedings of the 26th Australasian Computer Science Conference. 16*, pp. 255-264. Australian Computer Society, Inc.

Tsukamoto, H., Nagumo, H., Takemura, Y., & Nitta, N. (2012). Change of Students' Motivation in an Introductory Programming Course for Non-computing Majors. *2012 IEEE 12th International Conference on Advanced Learning Technologies* (pp. 124-125). Rome, Italy: IEEE Computer Society. doi:10.1109/ICALT.2012.38

Tu, C.-H., McIsaac, M. S., Sujo-Montes, L. E., & Armfield, S. (2016). Building Mobile Social Presence for U-Learning. Em *Human-Computer Interaction: Concepts, Methodologies, Tools, and Applications.* USA: Information Resources Management Association. doi:10.4018/978-1-4666-8789-9.ch004

Tu, C.-H., Sujo-Montes, L. E., & Yen, C.-J. (2015). Gamification for learning. Em R. Papa, *Media Rich Instruction* (pp. 203-217). Springer International Publishing. Obtido de https://doi.org/10.1007/978-3-319-00152-4_13

Tucker, A., & Noonan, R. (2006). *Programming Languages: Principles and Paradigms* (second ed.). (M.-H. Education, Ed.)

Turban, E., King, D., Lee, J. K., Liang, T. P., & Turban, D. C. (2015). *Electronic commerce: A managerial and social networks perspective.* Springer.

UNESCO. (2013). *UNESCO Policy guidelines for mobile learning.* Paris: United Nations Educational, Scientific and Cultural Organization. Obtido de https://unesdoc.unesco.org/ark:/48223/pf0000219641

UNESCO. (03 de 04 de 2018). *UNESCO Glossary.* Obtido de http://uis.unesco.org/en/glossary

United Nations. (2017a). *World Population Prospects: The 2017 Revision, Volume I: Comprehensive Tables (ST/ESA/SER.A/399).* United Nations, Department of Economic and Social Affairs, Population Division. Obtido de https://population.un.org/wpp/Publications/Files/WPP2017_Volume-I_Comprehensive-Tables.pdf

United Nations. (2017b). *World Population Prospects: The 2017 Revision, Volume II: Demographic Profiles (ST/ESA/SER.A/400).* United Nations, Department of Economic and Social Affairs, Population Division. Obtido de https://population.un.org/wpp/Publications/Files/WPP2017_Volume-II-Demographic-Profiles.pdf

Van Noy, M., James, H., & Bedley, C. (2016). *Reconceptualizing Learning: A Review of the Literature on Informal Learning.* ACT Foundation. Rutgers Education and Employment Research Center.

van Rossum, G. (1995). *Python Tutorial.* Amsterdam, The Netherlands: CWI - Centre for Mathematics and Computer Science.

Vasconcelos, J. B., & Carvalho, A. V. (2005). *Algoritmia e Estruturas de Dados - Programação nas Linguagens C e Java.* Edições Centro Atlântico.

Verdú, E., Regueras, L. M., Verdú, M. J., Leal, J. P., Castro, J. P., & Queirós, R. (2012). A distributed system for learning programming on-line. *Computers & Education, 58*, 1-10.

Viegas, M. C., Alves, G. R., Marques, M. A., Lima, N., Felgueiras, M. C., Costa, R. J., . . . Kreiter, C. (2017). VISIR+ Project – Preliminary results of the training actions. *14th Remote Engineering and Virtual Instrumentation (REV) Conference.* New York, USA.

von Rosing, M., Scheer, A.-W., & von Scheel, H. (2015). *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM* (first ed., Vol. 1). San Francisco, California, USA: Elsevier Science.

VPL - the Virtual Programming Lab for Moodle. (n.d.). Retrieved 01 31, 2018, from http://vpl.dis.ulpgc.es

Waite, M. (1992). *The Waite Group's Visual Basic 6 How-To.*

Wanhenheim, A. V., Martina, J. E., Cancian, R. L., & Dovichi, J. C. (2015). *Developing Programming Courses with Moodle and VPL - The Teacher's Guide to the Virtual Programming Lab.* Bookess. doi:10.13140/RG.2.1.1655.8165

Warren, D., Pereira, L. M., & Pereira , F. (1977). The Language and Its Implementation Compared with LISP. *Proceedings of SIGART/SIGPLAN Conference on Programming Languages.* Rochester, New York, USA.

Wasik, S., Antczak, M., Laskowski, A., & Sternal, T. (2018). A Survey on Online Judge Systems and Their Applications. *ACM Computing Surveys , 51*(1), 34. doi:10.1145/3143560

Williams, M. R. (1985). *A history of computing technology.* Upper Saddle River, New Jersey: Prentice-Hall, Inc. Obtido em 15 de 12 de 2017

Wirth, N. E. (1971). The programming language Pascal. *Acta Informatica, 1*, pp. 35-63.

Wirth, N. E. (1996). Recollections about the development of Pascal. Em T. J. Bergin Jr., & R. G. Gibson Jr, *History of programming languages---II* (pp. 97-120). New York, USA: ACM. doi:10.1145/234286.1057812

Zazpe, P. R. (2017). Los sistemas multimedia en la formación de documentalistas: un prototipo de entorno digital de aprendizaje aplicado a la informática documental. *Tese de Doutoramento*. Universidad Complutense de Madrid - Facultad de Ciencias de la Información.

Annexes

Annex A – Request for installation of the VPL in ISEP's Moodle

Nº Requerimento: ISEP - 33551 - 2017-07-28 14:47

Requerente: José Marilio Oliveira Cardoso (JOC - 15893)
Depto/Serviço: Engenharia Informática
Assunto: Pedido de instalação de plugin no Moodle institucional.
Estado: **Concluído**
Requerimento:
Exmo. Senhor Presidente do ISEP,

No âmbito do meu trabalho de Doutoramento pretendo realizar um estudo que implica o uso de um laboratório virtual de programação.
Pretendo que o referido estudo seja realizado na UC de APROG da LEI, no próximo semestre letivo.
Neste sentido, venho por este meio solicitar a V. Exa a instalação do plugin VPL (Virtual Programming Lab) no Moodle institucional do ISEP, com a brevidade possível e por um período de, pelo menos, três anos.
Solicito ainda que seja dado acesso ao referido plugin aos docentes MCN, JOC, AVC e RGB.

Pede deferimento,

Marílio Cardoso

Pareceres
De: Horácio José Almeida Macedo (HJAM)
Data: 2017-08-01 13:20
Parecer:
Em princípio sim, mas terá de ser testado antes.

Decisão de: José Carlos Barros Oliveira (JBO)
Data : 2017-08-02 12:30
Decisão: **Deferido**

Annex B – Request for authorisation from the Director of the LEI

De: Angelo Martins
Para: Marílio Cardoso
Cc: António Castro; Maria C. Neves
Assunto: RE: Experiência VPL em APROG
Data: 3 de agosto de 2017 12:55:40

Boa tarde,

Nada tenho a opor quanto ao uso de Laboratórios Virtuais de Programação em APROG. A responsabilidade pelo seu enquadramento no processo de aprendizagem da UC é do RUC (em CC). Lembro apenas que no próximo ano APROG passará para uma abordagem "early objects".

Cumprimentos,
Angelo Martins

From: Marílio Cardoso
Sent: 01 August 2017 12:16
To: Angelo Martins <amm@isep.ipp.pt>
Cc: António Castro <avc@isep.ipp.pt>
Subject: Experiência VPL em APROG
Importance: High

Exmo. Senhor Prof. Doutor Ângelo Martins

Diretor da Licenciatura em Engenharia Informática

Conforme conversado anteriormente com o Professor António Castro, serve o presente para o informar que no primeiro semestre, se irá proceder a uma experiência sobre o uso de Laboratórios Virtuais de Programação, no ensino de programação.
Para o efeito levaremos a cabo a investigação no âmbito da Unidade Curricular de APROG.
Já foi solicitado aos serviços centrais do ISEP a instalação do plugin VPL no Moodle institucional sendo nesta fase de acesso restrito a alguns docentes envolvidos diretamente no estudo.
De salientar que esta investigação não altera o planeamento pedagógico existente da UC.

Com o objetivo de documentar o processo, venho por este meio solicitar-lhe o favor de confirmar a sua anuência em relação à realização da referida experiência.

Com os melhores cumprimentos,

Marílio Cardoso

Annex C – Request for authorisation from the APROG Head

De:	Maria C. Neves
Para:	Marílio Cardoso
Cc:	António Castro
Assunto:	RE: Experiência VPL em APROG
Data:	1 de agosto de 2017 19:04:57

Boa tarde,

Conforme já falado anteriormente concordo com a experiência a realizar sobre o uso de Laboratórios Virtuais de Programação em algumas turmas da unidade curricular de APROG.

Cumprimentos

Maria da Conceição Neves
Adjunct Professor
Informatics Department
ISEP - School of Engineering
IPP - Polytechnic Instituto of Porto
mcn@isep.ipp.pt

De: Marílio Cardoso
Enviada: 1 de agosto de 2017 12:18
Para: Maria C. Neves
Cc: António Castro
Assunto: Experiência VPL em APROG
Importância: Alta

Exma. Senhora Prof. Doutora Conceição Neves

Responsável pela unidade curricular de Algoritmia e Programação

Conforme conversado anteriormente serve o presente para a informar que no primeiro semestre, se irá proceder a uma experiência sobre o uso de Laboratórios Virtuais de Programação, no ensino de programação.
Para o efeito levaremos a cabo a investigação no âmbito da Unidade Curricular de APROG.
Já foi solicitado aos serviços centrais do ISEP a instalação do plugin VPL no Moodle institucional sendo nesta fase de acesso restrito a alguns docentes envolvidos diretamente no estudo.
De salientar que esta investigação não altera o planeamento pedagógico existente da UC.

Com o objetivo de documentar o processo, venho por este meio solicitar-lhe o favor de confirmar a sua anuência em relação à realização da referida experiência.

Com os melhores cumprimentos,

Marílio Cardoso

Annex D – Curricular Unit sheet for APROG - 2017/2018

F10. FICHA DE UNIDADE CURRICULAR - R5/R6 (../EDUCATION/PREENCHE_FICHA_UC_V6.ASPX?CDE=20556)

Ficha UC

I - IDENTIFICAÇÃO / IDENTIFICATION

INSTITUIÇÃO / INSTITUTION: Instituto Superior de Engenharia do Porto

CURSO / DEGREE: Licenciatura em Engenharia Informática

UNIDADE CURRICULAR / COURSE TITLE: Algoritmia e Programação - APROG

ANO ESCOLAR / ACADEMIC YEAR: 2017-2018

ANO / YEAR	SEMESTRE / SEMESTER	HORAS-SEMANA / HOURS-WEEK *	ECTS
1	1º Semestre	T: 1; TP: 1; PL: 4	6

* T: TEÓRICA / LECTURE; TP: TEÓRICO-PRÁTICA / PRACTICAL; PL: PRÁTICA LABORATORIAL / LABORATORY; TC: TRABALHO DE CAMPO / FIELD WORK; OT: ORIENTAÇÃO TUTORIAL / TUTORIAL

PRÉ-REQUISITOS FORMAIS (PRECEDÊNCIAS) / FORMAL PRECEDENCES:

PORTUGUÊS	ENGLISH
Não tem	None

DOCENTES / PROFESSORS

	NOME / NAME	SIGLA / ACRONYM	HABILITAÇÕES / QUALIFICATIONS
RESPONSÁVEL / RESPONSIBLE	Maria da Conceição Carvalho Benta de Oliveira Neves	MCN	Doutoramento
OUTROS / OTHERS	Antonio Abel Vieira De Castro	AVC	Doutoramento
	António Alexandre De Sousa Gouveia	AAS	Mestrado
	Fernando Jorge Ferreira Duarte	FJD	Doutoramento
	José Marilio Oliveira Cardoso	JOC	Licenciatura
	Nuno Miguel Vieira Morgado	NVM	Licenciatura
	Paula Correia Tavares	PCT	Licenciatura
	Ricardo Gabriel Soares Fernandes De Almeida	RAL	Mestrado
	Rosa Cristina Gonçalves Barroso	RGB	Mestrado
	Rui Filipe Nogueira Marques	RFM	Mestrado

II - PROPÓSITOS, RESUMO, CARATERIZAÇÃO / PURPOSES, OVERVIEW, DESCRIPTION

ENQUADRAMENTO / FRAMEWORK

PORTUGUÊS

Programar é uma competência essencial para os estudantes de Engenharia Informática. Nesta unidade curricular são introduzidos os conceitos fundamentais associados à lógica da programação segundo o paradigma de programação procedimental, focando-se essencialmente na modelação algorítmica de soluções de problemas. A implementação dos algoritmos concebidos será feita utilizando a linguagem Java que permitirá fazer uma mais rápida evolução (em unidades curriculares posteriores) do paradigma procedimental para o paradigma de programação orientado aos objectos

ENGLISH

Programming is an essential skill for all Informatics Engineering students. In this course are introduced the fundamental concepts to programming under the paradigm of procedural programming, focusing primarily on designing algorithmic solutions to model computational processes associated to problems resolution. The implementation of the designed algorithms will be done using the Java language in order to make a faster evolution (in later courses) from the procedural paradigm to the object-oriented paradigm.

CONHECIMENTOS PRÉVIOS ASSUMIDAMENTE ADQUIRIDOS / REQUIRED PREVIOUS KNOWLEDGE

PORTUGUÊS

Não requer requisitos adicionais

ENGLISH

None additional requirement.

PROPÓSITOS E OBJETIVOS / PURPOSES AND OBJECTIVES

PORTUGUÊS

- Introdução de conceitos básicos das Ciências da Computação associados à Programação
- Analise e resolução de problemas computacionalmente com foco na modelação algorítmica.
- Concepção de algoritmos aplicando adequadas metodologias de programação
- Codificação de algoritmos em linguagem Java (essencialmente na perspectiva procedimental).
- Testar apropriadamente os programas
- Aplicar o que aprendemà resolução de problemas do mundo real
- Promover atitudes de aprendizagem activa, colaborativa e responsável, de trabalho persistente e de aplicação de espírito crítico na análise e resolução de problemas.
- Falar e escrever acerca do que aprendem.

O aluno deverá ser capaz de:
1-Compreender e aplicar os conceitos fundamentais da programação (Ponto 1 do programa)
2- Identificar os requisitos de um problema, analisá-lo, criar um algoritmo para a sua solução computacional e conceber um plano de testes apropriados para a sua validação. (Pontos 1,2,e 3 do programa)
3- Conhecer e entender a linguagem de programação Java na perspetiva essencialmente processual e aplicá-la a Implementação 'algoritmos. Testando as soluções usando o plano de teste apropriado. (Pontos 2,e 3 do programa)
4- Analisar e projetar algoritmos como modelos de processos computacionais estruturados em módulos, criar e reutilizar módulos e implementar em Java. (Pontos: 3 e 4 do programa).
5- Conhecer, compreender e utilizar estruturas de dados indexados, bem como manipular arquivos de texto. (Ponto 5 do programa)
6-Aplicar os conhecimentos adquiridos para resolver problemas reais e trabalhar cooperativamente na solução de problemas e fazer análise critica.(Todos os conteúdos)

ENGLISH

- Introduction of basic concepts related to Computer Science Programming
- Analyzing and solving problems computationally focusing on algorithmic modeling
- Designing algorithms by applying appropriate programming methodologies
- Codification of algorithms in Java language (mainly in the procedural paradigm based on classes)
- Testing appropriately the programs
- Apply what learn to solve real-world problems
- Promoting attitude of active learning, persistent work and also application of critical analysis in problem resolution
- Talk and write about what they learn.

The student should be able to:
1- Understand and apply the fundamental programming concepts. Syllabus point related to this purpose: 1.
2- Identify and describe the requirements of a problem, analyzing it, design an algorithm for its computational solution and conceive an appropriate tests? plan for its validation. . Syllabus points related to this purpose: 1,2 and 3.
3- Know and understand Java programming language in the perspective essentially procedural and apply it to algorithms? Implementation. Testing the solutions using the appropriate test plan. Syllabus points related to this purpose: 2,3
4- Analyse and designing algorithms as models of computational processes structured in modules, creating and reusing modules and implement them. Syllabus points related to this purpose:3, 4
5- Know, understand and use indexed data structures as well as manipulate text files. Syllabus points related to this purpose: 5
6- Apply the acquired knowledge to solve real problem (All syllabus).

PROGRAMA / PROGRAMME

PORTUGUÊS	ENGLISH
1. Conceitos básicos de Programação (20%) 1.1 Resolução de Problemas 1.2 Programação Procedimental Estruturada 1.3 Algoritmos e Programas 1.4 Variáveis, Tipos de Dados, Expressões e Operadores 1.5 Estruturas de Controlo de Fluxo 1.6 Descrição de algoritmos: Pseudo-Código e Fluxogramas 2. Codificação de programas (15%) 2.1 Linguagens de Programação -linguagem Java 2.2 Ambientes de desenvolvimento 2.3 Estrutura de um programa Java 2.4 Tipos de Dados e Variáveis 2.5 Operadores e Expressões e Atribuições 2.6 Codificação das Estruturas de Controlo de Fluxo 3. Tipos de Dados Referência: Classes (5%) 3.1 Classes e Objectos - APIs do Java 4. Decomposição Modular (10%) 4.1 Métodos e passagem de parâmetros 5. Tipos de Dados Referência: Arrays (20%) 5.1 Manipulação de Estruturas de Dados Indexadas 5.2 Algoritmos de procura, ordenação e de fusão ("merge") 6. Manipulação de Ficheiros de texto (5%) 7. Desenvolvimento de aplicações (25%)	1. Programming Fundamentals (20%) 1.1 Computational Resolution of Problems 1.2 Procedural Structured Programming 1.3 Algorithms and Programs 1.4 Variables, Data Types, Expressions and Operators 1.5 Flow Control Structures 1.6 Description of algorithms: Pseudo-Code and Flowcharts 2. Programs' codification (15%) 2.1 Programming Languages-Java Language 2.2 Integrated Development Environments 2.3 Java Program Structure 2.4 Data Types and Variables 2.5 Operators and Expressions and Assignments 2.6 Coding of Flow Control Structures 3. Data Types Reference: Classes (5%) 3.1 Classes and Objects 3.2 Java APIs 4. Modular Decomposition (10%) 4.1 Methods and parameters 5. Data Types Reference : Arrays (20%) 5.1 Manipulating Arrays mono and bi-dimensionals 5.2 Search, sort and merge algorithms 6. Text Files Manipulation (5%) 7.Developing applications (25%)

MATERIAL E FERRAMENTAS DE ENSINO MAIS IMPORTANTE / MOST IMPORTANT STUDING MATERIAL AND TOOLS

PORTUGUÊS	ENGLISH
-Página Web de APROG - https://moodle.isep.ipp.pt/ -Tópicos das Aulas Teóricas- Maria da Conceição Neves -Big Java -Late Objects - Cay S. Horstmann - John Wiley & Sons, Inc	-APROG Web site - https://moodle.isep.ipp.pt/ -Tópicos das Aulas Teóricas- Maria da Conceição Neves -Big Java -Late Objects - Cay S. Horstmann - John Wiley & Sons, Inc

MATERIAL DE ENSINO COMPLEMENTAR / SUPPLEMENTARY STUDING MATERIAL

PORTUGUÊS	ENGLISH
- Java: How to Program (How to Program Series)? - Deitel & Deitel ? Prentice Hall - informação suplementar - http://www.oracle.com/technetwork/java/	- Java: How to Program (How to Program Series)? - Deitel & Deitel ? Prentice Hall - Supplementary information - http://www.oracle.com/technetwork/java/

METODOLOGIA ENSINO-APRENDIZAGEM / TEACHING-LEARNING METHODOLOGY

PORTUGUÊS	ENGLISH
Nas aulas teóricas são apresentados, analisados e discutidos conceitos fundamentais associados à resolução de problemas computacionalmente e à linguagem Java. Nas aulas teórico-práticas são usadas técnicas da aprendizagem activa/cooperativa com trabalho em grupo e aprendizagem baseada em problemas. Nas aulas práticas laboratoriais os estudantes, em grupo, resolvem problemas, testam as soluções e discutem-nas com o professor. Os problemas estão organizados em 3 blocos com objetivos próprios	Lectures - are presented, analysed and discussed fundamental concepts associated with computational problem solving and also to the Java programming language. Practical classes - use techniques of active learning, cooperative group work and problem-based learning. Lab classes- use techniques of active learning.The students solve, in group, concrete problems, test and discuss them with the teacher receiving feedback. The problems are organized in three blocks each one with specific purposes.

DISTRIBUIÇÃO PERCENTUAL ESTIMADA DOS CONTEÚDOS / ESTIMATED PERCENTAGE DISTRIBUTION OF THE CONTENTS

PORTUGUÊS / ENGLISH

COMPONENTE CIENTÍFICA / SCIENTIFIC COMPONENT	40
COMPONENTE TECNOLÓGICA / TECHNOLOGICAL COMPONENT	50
COMPONENTE CONTEXTO ENVOLVENTE / SURROUNDING CONTEXT COMPONENT	10

RESULTADOS EXPETÁVEIS / OUTCOMES

PORTUGUÊS	ENGLISH
Os estudantes devem ser capazes de: - conhecer, compreender e aplicar conceitos fundamentais da programação; - conhecer, compreender e aplicar o paradigma da programação procedimental-algoritmos e estruturas de dados; - conhecer, compreender e aplicar a metodologia da programação estruturada; - conhecer, compreender e aplicar a linguagem de programação Java na perspectiva essencialmente procedimental; - analisar, avaliar e conceber algoritmos como modelos de processos computacionais usando as estruturas de dados mais adequadas; - analisar e resolver problemas usando o paradigma procedimental estruturado em módulos; - programar com clareza usando as melhores práticas; - conceber adequados planos de testes. Os estudantes são também encorajados a: - aplicar o que aprendem para resolver problemas do mundo real; - falar e escrever acerca do que aprendem; - trabalhar cooperativamente na resolução de problemas e na análise crítica de soluções.	Students should be able to: - know, understand and apply fundamental concepts of programming; - know, understand and apply the procedural programming paradigm - algorithms and data structures; - know, understand and apply the structured programming methodology; - know, understand and apply Java programming language in the perspective essentially procedural; - analyze, evaluate and design algorithms as models of computational processes using the most appropriate data structures; - analyze and solve problems using the procedural paradigm structured in modules; - program clearly using the best practices; - conceive adequate tests' plan. Students are always encouraged to: - apply what they learn to solve real world problems; - talk and write about what learning. - work cooperatively in problem solving and in critical analysis of solutions.

III - PROCEDIMENTOS DE AVALIAÇÃO / EVALUATION PROCEDURES

TIPO DE AVALIAÇÃO / EVALUATION TYPE

AVALIAÇÃO DURANTE O PERÍODO LETIVO COM AVALIAÇÃO FINAL OBRIGATÓRIA / EVALUATION DURING THE TERM WITH MANDATORY FINAL EVALUATION

Os estudantes têm que realizar parte da avaliação antes do período de exames, sendo a restante avaliação realizada no período de exames se os mínimos indicados na FUC (caso existam) forem atingidos. A avaliação durante o período letivo não deverá ter um peso inferior a 30% da classificação final.

FREQUÊNCIA ANTERIOR ACEITE / PREVIOUS YEAR PARTIAL EVALUATION ACCEPTED Não / No

A CLASSIFICAÇÃO DE FREQUÊNCIA ANTERIOR SERÁ SEMPRE ACEITE DESDE QUE NÃO EXISTA ALTERAÇÃO DOS CRITÉRIOS DE AVALIAÇÃO.

NÚMERO DE MOMENTOS DE AVALIAÇÃO DURANTE O PERÍODO LETIVO / NUMBER OF EVALUATION MOMENTS DURING THE TERM 3

COMPONENTES / COMPONENTS	TIPO / TYPE	PESO / WEIGHT (%)	MÍN / MIN	REPETÍVEL NORMAL / REPEATABLE 1ST EXAM	REPETÍVEL RECURSO / REPEATABLE 2ND EXAM
M1 Bloco1	L	18	0		
M2 Bloco2	L	21	0		
M3 Bloco3	L	21	0		
EXAME / EXAM	E	40	8.00	---	---

T: PROVAS ESCRITAS (TESTES, QUESTÕES, ...) / WRITTEN TESTS;
L: TRABALHOS DE LABORATÓRIO OU DE CAMPO / LAB WORKS, OR FIELDWORKS;
P: PROJETO, ESTÁGIO, DISSERTAÇÃO / PROJECT, THESIS;
R: RELATÓRIOS, TRABALHOS DE PESQUISA, TRABALHOS APLICADOS / PAPERS, APPLIED WORKS, ACADEMIC WORKS;
A: APRESENTAÇÕES / PRESENTATIONS;
E: AVALIAÇÃO EM PERÍODO DE EXAMES (EXAMES OU OUTROS) / EVALUATION AT THE END OF THE TERM (EXAMS OR OTHERS);
CF: CLASSIFICAÇÃO FINAL / FINAL CLASSIFICATION

AVALIAÇÃO DURANTE O PERÍODO LETIVO E ÉPOCA NORMAL / EVALUATION DURING THE TERM AND 1ST EXAM

CF = M1*0.18 + M2*0.21 + M3*0.21 + E*0.4

OUTROS MÍNIMOS NORMAL / OTHER MINIMUMS 1ST EXAM: 8.0

AVALIAÇÃO ÉPOCA RECURSO / 2ND EXAM EVALUATION

CF = M1*0.18 + M2*0.21 + M3*0.21 + E*0.4

OUTROS MÍNIMOS RECURSO / OTHER MINIMUMS 2ND EXAM: 8.0

OBSERVAÇÕES / COMMENTS

PORTUGUÊS	ENGLISH
O aluno para poder ter acesso a qualquer das provas de exame tem que obter no conjunto dos três blocos de trabalhos, desenvolvidos durante o periodo letivo, pelo menos a classificação média de 8.0 valores. Não é permitido repetir qualquer destes blocos de trabalhos porque refletem a evolução do desempenho do estudante ao longo do semestre.	The student in order to have access to any of the exams must obtain in the set of three blocks of works, developed during the academic period, at least the average classification of 8.0 values. NOT ALLOWED repeat the submission of any of the assignments because they reflect the evolution of student performance throughout the semester.

MELHORIA DE NOTA DE AVALIAÇÃO

PORTUGUÊS	ENGLISH
A prova de exame será constituída por: - uma prova escrita do tipo da referida anteriormente (PE), e - uma prova prática no computador (esta prova pode ser substituída pela última nota de frequência obtida) Estas provas têm pesos respetivamente de 60% e 40% na nota final.	This exam will include : - A written proof of the type referred above (EXAM), and - A practical proof on the computer (This proof can be replaced by the last frequency mark refered as NFREQ) These proofs have weights respectively 60% and 40% of the final grade. The student must inform one week before the exam, the responsible of the discipline about his choice for the practical proof.

NOTA BIOGRÁFICA DO RESPONSÁVEL DA UNIDADE CURRICULAR / BIOGRAPHICAL NOTE OF RESPONSIBLE OF THE COURSE

PORTUGUÊS	ENGLISH
A docente responsável pela disciplina é Professora Adjunta do Departamento de Engenharia Informática. Tem como habilitações académicas a Licenciatura em Engenharia Electrotécnica opção- Sistemas Digitais e Computadores pela FEUP, o Mestrado em Informática pela Universidade do Minho e Doutoramento em Engenharia Electrotécnica e Computadores - Inteligência Artificial Distribuída e Robótica pela FEUP.	The responsible for this APROG course is Professor Adjunto of Department of Computer Science. Her academic qualifications are: Licenciatura in Electrical Engineering - Digital Systems and Computers at FEUP-Engineering Faculty of Porto University; Master in Computer Science at the University of Minho; PhD in Electrical and Computer Engineering- Distributed Artificial Intelligence and Robotics at FEUP -Engineering Faculty of Porto University.

ISEP-UAE-MOD001v01

Data/hora de aprovação da ficha de unidade curricular: 2017-09-18 10:55.

Annex E – Curricular Unit sheet for APROG - 2018/2019

F10. FICHA DE UNIDADE CURRICULAR - R5/R6 (../EDUCATION/VISUALIZA_FICHA_UC_V7.ASPX?CDE=21516)

Ficha UC

I - IDENTIFICAÇÃO / IDENTIFICATION

INSTITUIÇÃO / INSTITUTION: Instituto Superior de Engenharia do Porto

CURSO / DEGREE: Licenciatura em Engenharia Informática

UNIDADE CURRICULAR / COURSE TITLE: Algoritmia e Programação - APROG

ANO ESCOLAR / ACADEMIC YEAR: 2018-2019

ANO / YEAR	SEMESTRE / SEMESTER	HORAS-SEMANA / HOURS-WEEK *	ECTS
1	1º Semestre	T: 1; TP: 1; PL: 4	6

* T: TEÓRICA / LECTURE; TP: TEÓRICO-PRÁTICA / PRACTICAL; PL: PRÁTICA LABORATORIAL / LABORATORY; TC: TRABALHO DE CAMPO / FIELD WORK; OT: ORIENTAÇÃO TUTORIAL / TUTORIAL

PRÉ-REQUISITOS FORMAIS (PRECEDÊNCIAS) / FORMAL PRECEDENCES:

PORTUGUÊS	ENGLISH
Não tem	None

DOCENTES / PROFESSORS

	NOME / NAME	SIGLA / ACRONYM	HABILITAÇÕES / QUALIFICATIONS
RESPONSÁVEL / RESPONSIBLE	Maria da Conceição Carvalho Benta de Oliveira Neves	MCN	Doutoramento
OUTROS / OTHERS	António Alexandre De Sousa Gouveia	AAS	Mestrado
	José Marilio Oliveira Cardoso	JOC	Licenciatura
	Nuno Miguel Gomes Bettencourt	NMB	Doutoramento
	Paula Correia Tavares	PCT	Doutoramento
	Rui Filipe Nogueira Marques	RFM	Especialista

II - PROPÓSITOS, RESUMO, CARATERIZAÇÃO / PURPOSES, OVERVIEW, DESCRIPTION

ENQUADRAMENTO / FRAMEWORK

PORTUGUÊS

Programar é uma competência essencial para os estudantes de Engenharia Informática. Nesta unidade curricular são introduzidos os conceitos fundamentais associados à lógica da programação segundo o paradigma de programação procedimental, focando-se essencialmente na modelação algorítmica de soluções de problemas. A implementação dos algoritmos concebidos será feita utilizando a linguagem Java que permitirá fazer uma mais rápida evolução (em unidades curriculares posteriores) do paradigma procedimental para o paradigma de programação orientado aos objectos

ENGLISH

Programming is an essential skill for all Informatics Engineering students. In this course are introduced the fundamental concepts to programming under the paradigm of procedural programming, focusing primarily on designing algorithmic solutions to model computational processes associated to problems resolution. The implementation of the designed algorithms will be done using the Java language in order to make a faster evolution (in later courses) from the procedural paradigm to the object-oriented paradigm.

CONHECIMENTOS PRÉVIOS ASSUMIDAMENTE ADQUIRIDOS / REQUIRED PREVIOUS KNOWLEDGE

PORTUGUÊS	ENGLISH
Não requer requisitos adicionais	None additional requirement.

PROPÓSITOS E OBJETIVOS / PURPOSES AND OBJECTIVES

PORTUGUÊS

Esta unidade curricular tem como objetivos:
- Introdução de conceitos básicos das Ciências da Computação associados à Programação
- Análise e resolução de problemas computacionalmente com foco na modelação algorítmica.
- Conceção de algoritmos aplicando adequadas metodologias de programação
- Codificação de algoritmos em linguagem Java (essencialmente na perspetiva procedimental).
- Testar apropriadamente os programas
- Aplicar o que aprendem à resolução de problemas do mundo real
- Promover atitudes de aprendizagem ativa, colaborativa e responsável, de trabalho persistente e de aplicação de espírito crítico na análise e resolução de problemas.
- Falar e escrever acerca do que aprendem.

No final o estudante deverá ser capaz de:
1-Compreender e aplicar os conceitos fundamentais da programação
2- Identificar os requisitos de um problema, analisá-lo, criar um algoritmo para a sua solução computacional e conceber um plano de testes apropriados para a sua validação.
3- Conhecer e compreender a linguagem de programação Java na perspetiva essencialmente procedimental e aplicá-la na Implementação algoritmos. Testando as soluções usando o plano de teste apropriado.
4- Analisar e projetar algoritmos como modelos de processos computacionais estruturados em módulos, criar e reutilizar módulos e implementar em Java.
5- Conhecer, compreender e utilizar estruturas de dados indexados, bem como manipular arquivos de texto.
6-Aplicar os conhecimentos adquiridos para resolver problemas reais e trabalhar cooperativamente na solução de problemas e fazer análise critica.

ENGLISH

The purposes of this curricular unit is:
- Introduction of basic concepts related to Computer Science Programming
- Analyzing and solving problems computationally focusing on algorithmic modeling
- Designing algorithms by applying appropriate programming methodologies
- Codification of algorithms in Java language (mainly in the procedural paradigm based on classes)
- Testing appropriately the programs
- Apply what learn to solve real-world problems
- Promoting attitude of active learning, persistent work and also application of critical analysis in problem resolution
- Talk and write about what they learn.

At the end the student should be able to:
1- Understand and apply the fundamental programming concepts.
2- Identify and describe the requirements of a problem, analyzing it, design an algorithm for its computational solution and conceive an appropriate tests' plan for its validation.
3- Know and understand Java programming language in the perspective essentially procedural and apply it to algorithms' Implementation. Testing the solutions using the appropriate test plan.
4- Analyse and designing algorithms as models of computational processes structured in modules, creating and reusing modules and implement them.
5- Know, understand and use indexed data structures as well as manipulate text files.
6- Apply the acquired knowledge to solve real problem.

PROGRAMA / PROGRAMME

PORTUGUÊS

1. Conceitos básicos de Programação (20%)
1.1 Resolução de Problemas
1.2 Programação Procedimental Estruturada
1.3 Algoritmos e Programas
1.4 Variáveis, Tipos de Dados, Expressões e Operadores
1.5 Estruturas de Controlo de Fluxo
1.6 Descrição de algoritmos: Pseudo-Código e Fluxogramas
2. Codificação de programas (15%)
2.1 Linguagens de Programação -linguagem Java
2.2 Ambientes de desenvolvimento
2.3 Estrutura de um programa Java
2.4 Tipos de Dados e Variáveis
2.5 Operadores e Expressões e Atribuições
2.6 Codificação das Estruturas de Controlo de Fluxo
3. Tipos de Dados Referência: Classes (5%)
3.1 Classes e Objectos - APIs do Java
4. Decomposição Modular (10%)
4.1 Métodos e passagem de parâmetros
5. Tipos de Dados Referência: Arrays (20%)
5.1 Manipulação de Estruturas de Dados Indexadas
5.2 Algoritmos de procura, ordenação e de fusão ("merge")
6. Manipulação de Ficheiros de texto (5%)
7. Desenvolvimento de aplicações (25%)

ENGLISH

1. Programming Fundamentals (20%)
1.1 Computational Resolution of Problems
1.2 Procedural Structured Programming
1.3 Algorithms and Programs
1.4 Variables, Data Types, Expressions and Operators
1.5 Flow Control Structures
1.6 Description of algorithms: Pseudo-Code and Flowcharts
2. Programs' codification (15%)
2.1 Programming Languages-Java Language
2.2 Integrated Development Environments
2.3 Java Program Structure
2.4 Data Types and Variables
2.5 Operators and Expressions and Assignments
2.6 Coding of Flow Control Structures
3. Data Types Reference: Classes (5%)
3.1 Classes and Objects
3.2 Java APIs
4. Modular Decomposition (10%)
4.1 Methods and parameters
5. Data Types Reference : Arrays (20%)
5.1 Manipulating Arrays mono and bi-dimensionals
5.2 Search, sort and merge algorithms
6. Text Files Manipulation (5%)
7.Developing applications (25%)

MATERIAL E FERRAMENTAS DE ENSINO MAIS IMPORTANTE / MOST IMPORTANT STUDING MATERIAL AND TOOLS

PORTUGUÊS

-Página Web de APROG - https://moodle.isep.ipp.pt/
-Tópicos das Aulas Teóricas- Maria da Conceição Neves
-Big Java -Late Objects - Cay S. Horstmann - John Wiley & Sons, Inc

ENGLISH

-APROG Web site - https://moodle.isep.ipp.pt/
-Tópicos das Aulas Teóricas- Maria da Conceição Neves
-Big Java -Late Objects - Cay S. Horstmann - John Wiley & Sons, Inc

MATERIAL DE ENSINO COMPLEMENTAR / SUPPLEMENTARY STUDING MATERIAL

PORTUGUÊS

- Java: How to Program (How to Program Series) - Deitel & Deitel - Prentice Hall
- informação suplementar - http://www.oracle.com/technetwork/java/

ENGLISH

- Java: How to Program (How to Program Series) - Deitel & Deitel -Prentice Hall
- Supplementary information - http://www.oracle.com/technetwork/java/

METODOLOGIA ENSINO-APRENDIZAGEM / TEACHING-LEARNING METHODOLOGY

PORTUGUÊS

Nas aulas teóricas são apresentados, analisados e discutidos conceitos fundamentais associados à resolução de problemas computacionalmente e à linguagem Java.
Nas aulas teórico-práticas são usadas técnicas da aprendizagem activa baseada em problemas.
Nas aulas práticas laboratoriais os estudantes, em grupo, resolvem problemas, testam as soluções e discutem-nas com o professor. Os problemas estão organizados em três blocos com objetivos próprios

ENGLISH

Lectures - are presented, analysed and discussed fundamental concepts associated with computational problem solving and also to the Java programming language.
Practical classes - use techniques of active and problem-based learning.
Lab classes- use techniques of active learning.The students solve, in group, concrete problems, test and discuss them with the teacher receiving feedback. The problems are organized in three blocks each one with specific purposes.

DISTRIBUIÇÃO PERCENTUAL ESTIMADA DOS CONTEÚDOS / ESTIMATED PERCENTAGE DISTRIBUTION OF THE CONTENTS

PORTUGUÊS / ENGLISH

COMPONENTE CIENTÍFICA / SCIENTIFIC COMPONENT	40
COMPONENTE TECNOLÓGICA / TECHNOLOGICAL COMPONENT	50
COMPONENTE CONTEXTO ENVOLVENTE / SURROUNDING CONTEXT COMPONENT	10

RESULTADOS EXPETÁVEIS / OUTCOMES

PORTUGUÊS

Os estudantes devem ser capazes de:
(# Conhecer e Compreender)
- conhecer e compreender conceitos fundamentais da programação;
- conhecer e compreender o paradigma da programação procedimental-algoritmos e estruturas de dados;
- conhecer e compreender a metodologia da programação estruturada;
- conhecer e compreender a linguagem de programação Java na perspectiva essencialmente procedimental;

(# Análise em Engenharia)
-Identificar e descrever os requisitos de um problema, analisá-lo para a sua solução computacional e conceber um plano de testes apropriados para a sua validação.

(# Design em Engenharia)
- Conceber algoritmos para a solução computacional de problemas bem como conceber um plano de testes apropriados para a sua validação.
- analisar, avaliar e conceber algoritmos como modelos de processos computacionais usando as estruturas de dados mais adequadas;

(# Prática em Engenharia)
- analisar e resolver problemas usando o paradigma procedimental estruturado em módulos;
- programar com clareza usando as melhores práticas;
- conceber adequados planos de testes.
- aplicar o que aprendem para resolver problemas do mundo real;

(#Comunicar e Trabalhar em equipa)
- falar e escrever acerca do que aprendem;
- trabalhar cooperativamente na resolução de problemas e na análise crítica de soluções.

ENGLISH

Students should be able to:
(#Knowledge and Understanding)
- know and understand fundamental concepts of programming;
- know and understand the procedural programming paradigm - algorithms and data structures;
- know and understand the structured programming methodology;
- know and understand Java programming language in the perspective essentially procedural;

(#Engineering Analysis)
- Identify and describe the requirements of a problem, analyzing it in order to conceive a computational solution and also to find an appropriate tests' plan for its validation.

(#Engineering Design)
- Design algorithms for problems' computational solution as well as conceive an appropriate tests' plan for its validation.
- analyze, evaluate and design algorithms as models of computational processes using the most appropriate data structures;

(#Engineering Practice)
- analyze and solve problems using the procedural paradigm structured in modules;
- program clearly using the best practices;
- conceive adequate tests' plan.
- apply what they learn to solve real world problems;

(#Communication and Team work)
- talk and write about what learning.
- work cooperatively in problem solving and in critical analysis of solutions.

III - PROCEDIMENTOS DE AVALIAÇÃO / EVALUATION PROCEDURES

TIPO DE AVALIAÇÃO / EVALUATION TYPE

AVALIAÇÃO DURANTE O PERÍODO LETIVO COM AVALIAÇÃO FINAL OBRIGATÓRIA

Os estudantes têm que realizar parte da avaliação antes do período de exames, sendo a restante avaliação realizada no período de exames se os mínimos indicados na FUC (caso existam) forem atingidos. A avaliação durante o período letivo não deverá ter um peso inferior a 30% da classificação final.
Students must complete part of the evaluation before the end of the term, being the rest carried out during the exam period provided that the minimum scores (if defined) listed in the Curricular Unit's Form are achieved. Evaluation during the term should not account for less than 30% of the final grade.

OS MOMENTOS DE AVALIAÇÃO NÃO REPETÍVEIS MANTÊM A PONDERAÇÃO E O CONTEÚDO PROGRAMÁTICO / NON REPEATABLE EVALUATION MOMENTS KEEP THE SAME WEIGHT AND CONTENT:

Não / No

OS ESTUDANTES PODERÃO SEMPRE MANTER A CLASSIFICAÇÃO DOS MOMENTOS DE AVALIAÇÃO NÃO REPETÍVEIS DESDE QUE NÃO EXISTA ALTERAÇÃO DOS CRITÉRIOS DE AVALIAÇÃO.

AVALIAÇÃO DURANTE O PERÍODO LETIVO / EVALUATION DURING THE TERM

NÚMERO DE MOMENTOS NÃO REPETÍVEIS (NR) / NUMBER OF NON REPEATABLE EVALUATION MOMENTS (NR)	3
NÚMERO DE MOMENTOS REPETÍVEIS (R) / NUMBER OF REPEATABLE EVALUATION MOMENTS (R)	0

COMPONENTES / COMPONENTS		TIPO / TYPE	PESO / WEIGHT (%)	MÍN / MIN
M1 - NR	Bloco1	L	18	0
M2 - NR	Bloco2	L	21	0
M3 - NR	Bloco3	L	21	0
MÉDIA PONDERADA MOMENTOS NÃO REPETÍVEIS / WEIGHED AVERAGE NON REPEATABLE MOMENTS			60	8.00
EXAME GLOBAL / GLOBAL EXAM		EG	40	8.0

T: PROVAS ESCRITAS (TESTES, QUESTÕES, ...) / WRITTEN TESTS;
L: TRABALHOS DE LABORATÓRIO OU DE CAMPO / LAB WORKS, OR FIELDWORKS;
P: PROJETO, ESTÁGIO, DISSERTAÇÃO / PROJECT, THESIS;
R: RELATÓRIOS, TRABALHOS DE PESQUISA, TRABALHOS APLICADOS / PAPERS, APPLIED WORKS, ACADEMIC WORKS;
A: APRESENTAÇÕES / PRESENTATIONS;
EG: AVALIAÇÃO GLOBAL EM PERÍODO DE EXAMES (EXAMES OU OUTROS) / GLOBAL EVALUATION AT THE END OF THE TERM (EXAMS OR OTHERS);
CF: CLASSIFICAÇÃO FINAL / FINAL CLASSIFICATION

FÓRMULA(S) DE CÁLCULO DA CLASSIFICAÇÃO FINAL / FINAL CLASSIFICATION FORMULA(S) (CF)

CF = M1*0.18 + M2*0.21 + M3*0.21 + EG*0.4

OBSERVAÇÕES / COMMENTS

PORTUGUÊS

O aluno para poder ter acesso a qualquer das provas de exame tem que obter no conjunto dos três blocos de trabalhos, desenvolvidos durante o periodo letivo, pelo menos a classificação média de 8.0 valores.
Não é permitido repetir qualquer destes blocos de trabalhos porque refletem a evolução do desempenho do estudante ao longo do semestre.

ENGLISH

The student in order to have access to any of the exams must obtain in the set of three blocks of works, developed during the academic period, at least the average classification of 8.0 values.
NOT ALLOWED repeat the submission of any of the assignments because they reflect the evolution of student performance throughout the semester.

MELHORIA DE NOTA DE AVALIAÇÃO

PORTUGUÊS

A prova de exame será constituída por:
- uma prova escrita do tipo da referida anteriormente (PE), e
- uma prova prática no computador
(esta prova pode ser substituída pela última nota de frequência obtida)
Estas provas têm pesos respetivamente de 60% e 40% na nota final.

ENGLISH

This exam will include :
- A written proof of the type referred above (EXAM), and
- A practical proof on the computer
(This proof can be replaced by the last frequency mark refered as NFREQ)
These proofs have weights respectively 60% and 40% of the final grade.
The student must inform one week before the exam, the responsible of the discipline about his choice for the practical proof.

NOTA BIOGRÁFICA DO RESPONSÁVEL DA UNIDADE CURRICULAR / BIOGRAPHICAL NOTE OF RESPONSIBLE OF THE COURSE

PORTUGUÊS

A docente responsável pela disciplina é Professora Adjunta do Departamento de Engenharia Informática. Tem como habilitações académicas a Licenciatura em Engenharia Electrotécnica opção- Sistemas Digitais e Computadores pela FEUP, o Mestrado em Informática pela Universidade do Minho e Doutoramento em Engenharia Electrotécnica e Computadores - Inteligência Artificial Distribuída e Robótica pela FEUP.

ENGLISH

The responsible for this APROG course is Professor Adjunto of Department of Computer Science. Her academic qualifications are: Licenciatura in Electrical Engineering - Digital Systems and Computers at FEUP-Engineering Faculty of Porto University; Master in Computer Science at the University of Minho; PhD in Electrical and Computer Engineering- Distributed Artificial Intelligence and Robotics at FEUP -Engineering Faculty of Porto University.

IV - INFORMAÇÃO PARA A3ES / INFORMATION FOR A3ES

DEMONSTRAÇÃO DA COERÊNCIA DOS CONTEÚDOS PROGRAMÁTICOS COM OS OBJETIVOS DA UNIDADE CURRICULAR / DEMONSTRATION OF THE SYLLABUS COHERENCE WITH THE CURRICULAR UNIT'S OBJECTIVES

PORTUGUÊS

(1.) Conceitos básicos das ciências da computação associados à programação - tem como objetivo dar ao aluno conhecimentos e competências para resolver computacionalmente problemas, identificando os requisitos e concebendo algoritmo adequado para a sua solução.
(2.) Codificação de programas e (3.) Clases e objetos - tem como objetivo dar ao aluno conhecimentos e competências para elaborar o programa codificado em Java e testar a solução utilizando um adequado plano de testes.
(4.) Decomposição Modular e (5.) Tipos de Dados Referência: Arrays e (6.) Manipulação de Ficheiros - tem como objetivo dar ao aluno conhecimentos e competências para resolver problemas mais complexos usando as estruturas de dados adequadas e estruturando-o em módulos, reutilizando os módulos existentes.
(7.) Desenvolvimento de aplicações - tem como objetivo dar ao aluno competências e desenvolver capacidades para integrar os conhecimentos, desenvolver soluções e emitir juízos de valor na realização e apresentação/discussão dos trabalhos

ENGLISH

(1.) Programming Fundamentals - aims to give the student knowledge and skills to solve problems computationally, identifying requirements and designing appropriate algorithm for its solution.
(2.) Programs' codification and (3.) Clases and objects - aims to give the student knowledge and skills to develop the program coded in Java and test the solution using an appropriate test plan.
(4.) Modular Decomposition and (5.) Reference Data Types: Arrays and (6.) Text Files Manipulation - aims to give the student knowledge and skills to solve more complex problems, using appropriates data structures, structuring it in modules and reusing existing modules.
(7.) Developing applications - aims to give the student develop skills and abilities to integrate knowledge, develop solutions and make value judgments on performance and presentation / discussion of the work

DEMONSTRAÇÃO DA COERÊNCIA DAS METODOLOGIAS DE ENSINO COM OS OBJECTIVOS DE APRENDIZAGEM DA UNIDADE CURRICULAR / DEMONSTRATION OF THE COHERENCE BETWEEN THE TEACHING METHODOLOGIES AND THE LEARNING OUTCOMES

PORTUGUÊS

Nas aulas teóricas são apresentados, analisados e discutidos conceitos fundamentais associados à resolução computacional de problemas e também à linguagem de programação em Java. Os conceitos e conteúdos apresentados são devidamente ancorados na sua aplicabilidade.
Nas aulas teórico-práticas e laboratoriais os alunos, trabalhando cooperativamente em grupo, resolvem os problemas propostos, adquirindo conhecimentos e competências. Os alunos aprendem a identificar os requisitos dos problemas, a conceber os algoritmo adequados para a sua solução, a elaborar programas codificados em Java e a testar as soluções utilizando adequados planos de testes. Ao longo das diversas atividades letivas o aluno é incentivado a adquirir conhecimentos e competências, a integrar adequadamente os conhecimentos, desenvolver soluções e emitir juízos de valor na realização e apresentação/discussão dos problemas.
A aquisição de conhecimentos e a avaliação são atividades que decorrem conjuntamente ao longo do período letivo com o objetivo de orientar o aluno na sua aprendizagem. O exame final tem como objetivo avaliar individualmente a integração do conhecimento e competências adquiridas.

ENGLISH

In the lectures are presented, analyzed and discussed fundamental concepts related to computational problems and also to the programming language Java. The concepts and content presented are properly anchored in their applicability.
In practical and laboratory classes the students, working cooperatively in groups, solve the proposed problems, acquiring knowledge and skills. They learn to identify requirements, design the appropriate algorithm for its solution, develop programs coded in Java and test the solution using appropriate test plans. Throughout the activities, the students are encouraged to acquire knowledge and skills, integrate appropriately the knowledge, develop solutions and make value judgments on performance and presentation / discussion of problems.
The knowledge acquisition and assessment are taking place together along the semester in order to guide the student in their learning. The purpose of the final exam is to assess individually the ability to integration of knowledge and skills.

ISEP-UAE-MOD001v01

Data/hora de aprovação da ficha de unidade curricular: 2018-09-15 21:33.

Annex F – Opinion survey – students 2017/2018

Inquérito VPL 2017-2018

Este formulário é anónimo e de preenchimento facultativo. É realizado no âmbito de um estudo sobre a integração de VPL (Virtual Programming Lab) no processo de ensino/aprendizagem de programação na UC de APROG.
Não existem respostas certas ou erradas. Por isso solicita-se a sua resposta espontânea e com a máxima sinceridade.
Em cada ponto deverá assinalar apenas UMA resposta.

* Required

1. **Género** *

Mark only one oval.

- Feminino
- Masculino

2. **Faixa etária** *

Mark only one oval.

- inferior a 20 anos
- entre 20 e 25 anos
- superior a 25 anos

3. **Antes de iniciar a frequência da unidade curricular de APROG, tinha experiência de programação?** *

Mark only one oval.

- Sim
- Não

4. **Quantidade de exercícios que submeti no VPL** *

Mark only one oval per row.

	0	1	2	3	4	5	6
-							

5. **Considere a sua perceção ou opinião quanto às afirmações, assinalando a resposta que corresponda ao seu grau de concordância.** *
Mark only one oval per row.

	Discordo totalmente	Discordo parcialmente	Não concordo nem discordo	Concordo parcialmente	Concordo totalmente
A programação de computadores é uma tarefa difícil					
Sou adepto do uso de plataformas digitais de ensino à distância (eLearning), tipo Moodle					
Gostaria de dispor de ferramentas de apoio à resolução de exercícios fora das aulas que desse feedback, mostrasse soluções e/ou guiasse a resolução					
A utilização do VPL na resolução de exercícios foi uma ajuda importante para o meu processo de aprendizagem					
Comecei por utilizar o VPL mas perdi o interesse					
O docente que promoveu a experiência explicou com clareza e deu apoio suficiente para a utilização eficaz do VPL					
A utilização do VPL é demasiado complicada					
Gostaria de ter tido a possibilidade de utilizar o VPL para a resolução de mais exercícios					

6. **Comentários/sugestões:**

Annex G – Opinion survey – students 2018/2019

Inquérito sobre usabilidade do VPL

Este inquérito é ANÓNIMO e de preenchimento facultativo.

É realizado no âmbito de um estudo sobre a integração de VPL (Virtual Programming Lab) no processo de ensino/aprendizagem de programação na Unidade Curricular de Algoritmia e Programação - APROG.

Não existem respostas certas ou erradas.
Solicita-se a sua resposta espontânea e com a máxima sinceridade.

* Required

1. **Género** *

   Mark only one oval.

   ◯ Feminino
   ◯ Masculino

2. **Faixa etária** *

   Mark only one oval.

   ◯ inferior a 20 anos
   ◯ entre 20 e 25 anos
   ◯ superior a 25 anos

3. **Antes de iniciar a frequência de APROG, tinha conhecimentos de algoritmia?** *

   Mark only one oval.

   ◯ Não
   ◯ Sim

4. **Antes de iniciar a frequência de APROG, tinha alguma experiência de programação?** *

   Mark only one oval.

   ◯ Não *Skip to question 8.*
   ◯ Sim *Skip to question 5.*

Experiência de programação

5. **Antes de iniciar a frequência de APROG, tinha experiência de programação em:** *
Check all that apply.

- [ ] Assembly
- [ ] C
- [ ] C++
- [ ] C#
- [ ] Java
- [ ] JavaScript
- [ ] MATLAB
- [ ] Pascal
- [ ] PHP
- [ ] Python
- [ ] Visual Basic
- [ ] Outras

6. **Na linguagem em que tem mais experiência de programação, essa experiência foi obtida em contexto** *
Check all that apply.

- [ ] Académico
- [ ] Laboral / profissional
- [ ] Autodidata / aprendizagem informal
- [ ] Lúdico

7. **Na linguagem em que tem mais experiência de programação, tem experiência de quantos anos?** *

____

Utilização do VPL

8. **Já tinha ouvido falar do VPL anteriormente?** *
Mark only one oval.

- ( ) Não
- ( ) Sim

9. **Já tinha usado o VPL anteriormente?** *
Mark only one oval.

- ( ) Não
- ( ) Sim

10. **Indique a quantidade de exercícios que submeteu no VPL** *
Mark only one oval per row.

	0	1	2	3	4	5	6
.	( )	( )	( )	( )	( )	( )	( )

11. **No caso de não ter submetido todos os exercícios indique as razões para o não ter feito.**

Utilização do VPL

12. **Dê a sua opinião relativamente a cada uma das afirmações seguintes:** *
Mark only one oval per row.

	Discordo totalmente	Discordo parcialmente	Não concordo nem discordo	Concordo parcialmente	Concordo totalmente
A programação é uma tarefa difícil					
As plataformas de ensino à distância (exemplo Moodle) são uma mais valia					
Gostaria de dispor de ferramentas de apoio à resolução de exercícios fora das aulas					
O exercício demonstrativo foi útil para a familiarização com o VPL					
A utilização do VPL na resolução de exercícios foi uma ajuda para o meu processo de aprendizagem					
Comecei por utilizar o VPL mas perdi o interesse					
A utilização do VPL é demasiado complicada					

Utilização do VPL (continuação)

13. **Dê a sua opinião relativamente a cada uma das afirmações seguintes:** *

Mark only one oval per row.

	Discordo totalmente	Discordo parcialmente	Não concordo nem discordo	Concordo parcialmente	Concordo totalmente
Gostaria de ter tido a possibilidade de utilizar o VPL para a resolução de mais exercícios	⬭	⬭	⬭	⬭	⬭
Gostaria de poder utilizar o VPL para efeitos de avaliação individual em substituição da resolução em papel	⬭	⬭	⬭	⬭	⬭
A possibilidade de resubmissões e obtenção de classificação automática é muito útil	⬭	⬭	⬭	⬭	⬭
Os testes pré-defiinidos foram importantes na identificação de deficiências nas minhas resoluções	⬭	⬭	⬭	⬭	⬭
O facto de o número de avaliações ser limitado foi um aspeto negativo para o meu trabalho	⬭	⬭	⬭	⬭	⬭
O VPL é uma mais valia para o processo de ensino / aprendizagem de programação	⬭	⬭	⬭	⬭	⬭
A experiência foi explicada com clareza	⬭	⬭	⬭	⬭	⬭
Foi dado apoio suficiente para a utilização eficaz do VPL	⬭	⬭	⬭	⬭	⬭

Apreciação do VPL

14. **Classifique cada uma das seguintes caraterísticas do VPL, de acordo com a sua opinião** *
Mark only one oval per row.

	Negativo	Neutro	Positivo
Utilização fora do tempo das aulas	◯	◯	◯
Editor incorporado (não IDE)	◯	◯	◯
Formatação rígida do output	◯	◯	◯
Obtenção de feedback sem intervenção do professor	◯	◯	◯
Testes pré-definidos	◯	◯	◯
Avaliação automática em função dos resultados	◯	◯	◯

15. **Indique outros aspetos do VPL que considere relevantes indicando, para cada um deles, se o considera positivo ou negativo.**

____
____
____
____
____

16. **Considera positivo o uso de VPL para aprendizagem de programação?** *
Mark only one oval.

◯ Sim
◯ Não

17. **Opinião geral sobre o VPL** *
Mark only one oval.

	1	2	3	4	5	
Mau	◯	◯	◯	◯	◯	Excelente

18. **Comentários/sugestões:**

____
____
____
____
____

Annex H – Opinion survey – teachers

Inquérito sobre utilização do VPL em APROG

Este inquérito é realizado no âmbito de um estudo sobre a integração de VPL (Virtual Programming Lab) no processo de ensino/aprendizagem de programação na Unidade Curricular de Algoritmia e Programação - APROG.

Peço a sua colaboração com respostas espontâneas e com a máxima sinceridade.

Obrigado.

* Required

1. **Email address** *

2. **Dê a sua opinião relativamente a cada uma das afirmações seguintes:** *
Mark only one oval per row.

	Discordo totalmente	Discordo parcialmente	Não concordo nem discordo	Concordo parcialmente	Concordo totalmente
Gostaria de ter tido a possibilidade de utilizar o VPL para a resolução de mais exercícios	⬭	⬭	⬭	⬭	⬭
Seria útil usar o VPL para efeitos de avaliação individual em substituição da resolução em papel	⬭	⬭	⬭	⬭	⬭
O VPL pode contribuir para a uniformização das notas da disciplina, tendo em consideração o volume de alunos, trabalhos, exercícios e diferentes avaliações levadas a cabo	⬭	⬭	⬭	⬭	⬭
O VPL pode diminuir o tempo que o professor usa no processo de ensino/aprendizagem e de avaliação	⬭	⬭	⬭	⬭	⬭
Comparativamente com outros anos letivos, a utilização do VPL teve um impacto positivo na aprendizagem	⬭	⬭	⬭	⬭	⬭
O feedback dado pelo VPL de forma automática e imediata é útil para o aluno	⬭	⬭	⬭	⬭	⬭
O VPL é uma mais valia para o processo de ensino / aprendizagem de programação	⬭	⬭	⬭	⬭	⬭

Apreciação do VPL

3. **Quanto tempo despende, em média, na "validação" de cada exercício de programação em Java das fichas de trabalho?** *

4. **Quanto tempo da aula despende, em média, na "validação" de exercícios de programação em Java das fichas de trabalho?** *

5. **Que alterações sugeriria, tanto do lado da disciplina como do lado do VPL, de forma a potenciar um maior sucesso na apreensão das matérias por parte dos alunos?** *

6. **Opinião geral sobre o VPL** *
   Mark only one oval.

	1	2	3	4	5	
Mau	⬭	⬭	⬭	⬭	⬭	Excelente

7. **Comentários/sugestões:**

Powered by
Google Forms

Annex I – CCPFC/ACC-101425/18 Accreditation Register

Ações de Formação c/despacho > Imprimir (id #102479)

Ficha da Ação

Título Programação Java

Área de Formação B - Prática pedagógica e didática na docência

Modalidade Curso de Formação

Regime de Frequência b-learning

Duração
Horas presenciais: 25
Nº de horas acreditadas: 25

Cód. Área **Descrição**

Cód. Dest. 99 **Descrição** Professores dos Grupos 500, 540 e 550
DCP 99 **Descrição** Professores dos Grupos 500, 540 e 550

Reg. de acreditação (ant.)

Formadores

Formadores com certificado de registo

B.I. 7311704 **Nome** José Marilio Oliveira Cardoso **Reg. Acr.** CCPFC/RFO-30034/11

Componentes do programa Introdução à linguagem Java **Nº de horas** 6

B.I. 8085612 **Nome** EMANUEL FERNANDO CUNHA SILVA **Reg. Acr.** CCPFC/RFO-26005/09

Componentes do programa Configuração de ferramenta de avaliação automática **Nº de horas** 6

Formadores sem certificado de registo

Conteúdos

Razões justificativas da ação e a sua inserção no plano de atividades da entidade proponente
Aprender a programar é um processo difícil para muitos estudantes, para o qual é necessário muito esforço, treino e motivação. Classificar o código desenvolvido e fornecer feedback imediato e significativo é um método popular e eficiente para envolver e motivar os estudantes de programação.
Esta ação visa dotar os formandos de conceitos fundamentais associados à lógica da programação focando-se na modelação algorítmica da solução de problemas e com implementação em Java.
Será ainda abordada e demonstrada a utilização de ferramentas de avaliação automática de programas de computador. É espectável que, com este tipo de ferramentas, os alunos sejam mais motivados e possam desenvolver competências de programação de forma mais célere e autónoma.

Objetivos a atingir
• Analisar resolver problemas computacionalmente
• Conhecer e entender a linguagem de programação Java (essencialmente na perspetiva procedimental)
• Codificar algoritmos em linguagem Java
• Promover atitudes de aprendizagem ativa e colaborativa, e de aplicação de espirito crítico na análise e resolução de problemas.
• Conhecer e utilizar ferramentas de avaliação automática de código
• Produzir exercícios e configurar ferramenta para avaliação automática de código

Conteúdos da ação
Módulo I – IDE NetBeans
1. Instalação
2. Editor NetBeans
3. Executar, Debbug
Módulo II – Introdução à Linguagem Java
1. Sintaxe elementar da linguagem

a) Tipos de dados
b) Estruturas de controlo de fluxo
2. Estrutura de um programa Java
3. Análise de exemplos ilustrativos de codificação Java
4. Conversão de algoritmos para Java
Módulo III – Ferramenta de avaliação automática / Programação em Java
1. Introdução à avaliação automática
2. Uso de avaliação automática
3. A Estruturas de repetição
4. Estruturas de dados Java
a. arrays
b. arraylists
5. Decomposição modular de funcionalidades
6. Análise de exemplos ilustrativos de codificação Java
7. Implementação de algoritmos em Java
8. Uso de avaliação automática

Módulo IV – Configuração de ferramenta de avaliação automática
1. Problemas
2. Testes
3. Linguagens de programação permitidas na resolução
4. Participantes

Metodologias de realização da ação
Sessões teóricas expositivas
Práticas Simuladas

Regime de avaliação dos formandos
Obrigatoriedade de frequência de 2/3 das horas presenciais.
Trabalhos práticos efetuados a partir das e nas sessões presenciais de acordo com os critérios previamente estabelecidos.
Participação na Formação (40%)
- participação ativa, interesse demonstrado, iniciativa, autonomia, assiduidade e pontualidade.
Explicitação formal de desempenho (60%)
- trabalho individual/grupo.

Bibliografia fundamental

Big Java -Late Objects - Cay S. Horstmann - John Wiley & Sons, Inc

Java: How to Program (How to Program Series)? - Deitel & Deitel ? Prentice Hall

Mooshak: a Web-based multi-site programming contest system - Software Practice and Experience 33(6):567-581 – May 2003

Formação a Distância

Demonstração das vantagens para os/as formandos/as no recurso ao regime de formação a distância
Flexibilidade no processo de ensino/aprendizagem – os formandos controlam a sua disponibilidade e o ritmo de aprendizagem de cada um deles é respeitado.
Custos reduzidos – pode frequentar o curso a partir de casa ou do trabalho necessitando apenas de um computador com ligação à Internet
Apoio do Formador – o e-formador desempenha um papel de facilitador da aprendizagem, fornecendo feedback das atividades e esclarecendo dúvidas no prazo máximo de 24 horas.

Distribuição de horas 12 **Nº de horas online síncrono** 0 **Nº de horas online assíncrono** 13

Demonstração da existência de uma equipa técnico-pedagógica que assegure o manuseamento das ferramentas e procedimentos do formação a distância
O apoio será dado pela divisão de Sistemas Informáticos - Gabinete de E-learning e Multimédia do ISEP e da unidade do E-IPP do IPP
A Unidade de e-Learning e Inovação Pedagógica do Politécnico do Porto visa contribuir para o desenvolvimento e implementação de novas metodologias e pedagogias no sentido de impulsionar a inovação nas formas de ensino/aprendizagem/formação na Comunidade P.PORTO e na sua área de influência.
e-ipp.ipp.pt/
O LCMS a ser usado será o moodle

Demonstração da implementação de um Sistema de Gestão da Aprendizagem / Learning Management System adequado
Existe um conjunto de plataforma que serão usados como o moodle já em funcionamento e usado nas licenciaturas e mestrados ministrados no ISEP
moodle.isep.ipp.pt

Demonstração da avaliação presencial (permitida a avaliação em videoconferência)
- Participação ativa, interesse demonstrado, iniciativa, autonomia, assiduidade e pontualidade;
- Trabalho individual/grupo.

Demonstração da distribuição da carga horária pelas diversas tarefas
Módulo I – IDE NetBeans (3h)
Módulo II – Introdução à Linguagem Java (6h)
Módulo III – Ferramenta de avaliação automática / Programação em Java (10h)
Módulo IV – Configuração de ferramenta de avaliação automática (6h)

Rácio de formadores/as por formandos/as 1

Processo

Data de receção 30-05-2018 **Nº processo** 102328 **Registo de acreditação** CCPFC/ACC-101425/18

Data do despacho 25-10-2018 **Nº ofício** 2889 **Data de validade** 25-10-2021

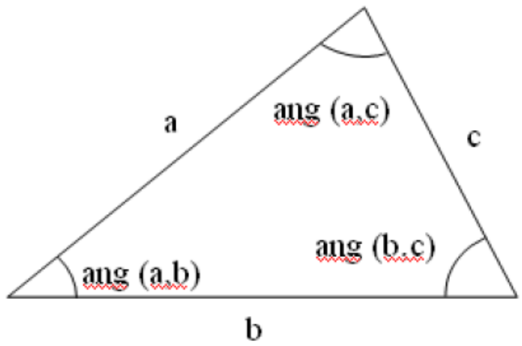
Estado do Processo C/ Despacho - Acreditado

Annex J – Exercises chosen to use with the VPL

PL6 – Exercício 3 (**)

a) Faça um método que calcule um ângulo interno de um triângulo sendo dadas as medidas dos três lados desse triângulo. O valor do ângulo deve estar em graus.

b) Sendo dadas as medidas de três lados, verifique se as medidas são válidas e se é possível formar um triângulo. Em caso afirmativo calcule todos os ângulos internos desse triângulo. Para isso chame três vezes o método desenvolvido na alínea anterior.



Ângulo	Fórmula
ang (a,b)	$arc\cos\left(\frac{a^2+b^2-c^2}{2ab}\right)$
ang (a,c)	$arc\cos\left(\frac{a^2+c^2-b^2}{2ac}\right)$
ang (b,c)	$arc\cos\left(\frac{b^2+c^2-a^2}{2bc}\right)$

PL6 – Exercício 5 (**)

a) Faça um módulo que dados dois números inteiros positivos retorne a quantidade de dígitos comuns nas mesmas posições.

b) Elabore um programa que leia N pares de valores inteiros positivos, sendo N introduzido pelo utilizador e validado. Após a leitura dos N pares de valores o programa deve apresentar o par que tiver mais dígitos comuns.

PL7 – Exercício 2 (**)

Pretende-se uma aplicação modular para determinar algumas estatísticas sobre vencimentos de funcionários duma empresa. O número de funcionários varia ao longo do tempo mas não é superior a 50.

O programa deve ter as seguintes funcionalidades:

a) Leitura de nomes e vencimentos de funcionários da empresa. A leitura deve terminar com a introdução do nome "tt";

b) Listagem dos nomes dos funcionários com vencimentos inferiores à média;

c) Apresentação da percentagem de funcionários com vencimentos inferiores a um dado valor fornecido pelo utilizador.

PL7 – Exercício 4 (***)

Elabore um programa modular que tenha as seguintes funcionalidades:

a) Leitura de N números inteiros para um vetor, sendo N definido pelo utilizador;

b) Inversão da ordem dos elementos do vetor;

Exemplo: | 2 | 3 | 4 | → | 4 | 3 | 2 |

c) Apresentação do vetor invertido;

d) Rotação para a direita dos elementos do vetor invertido;

Exemplo: | 4 | 3 | 2 | → | 2 | 4 | 3 |

e) Apresentação do vetor rodado.

PL8 – Exercício 2 (***)

Um quadrado mágico é uma matriz quadrada de números inteiros onde a soma dos números de qualquer linha, de qualquer coluna e das duas diagonais dá sempre o mesmo valor.

Exemplo de um quadrado mágico:

8	1	6
3	5	7
4	9	2

Elabore uma aplicação modular que permita verificar se uma dada matriz é um quadrado mágico.

PL8 – Exercício 3 (***)

Elabore um programa modular que permita ler uma matriz de números inteiros e determinar a frequência de ocorrência dos números na matriz.
Deverá ainda mostrar a matriz original bem como todos os números diferentes e respetiva frequência, ordenados de forma decrescente desta.

Exemplo:

Dada a matriz

8	-5	3	8
6	3	10	1
10	8	-5	2

Resultado:

8	-5	10	3	6	1	2
3	2	2	2	1	1	1

Annex K – Explanatory text of the use of VPL in 2019/2020

isep Instituto Superior de Engenharia do Porto

moodle Virtual Programming Lab

Utilização do Virtual Programming Lab (VPL) em APROG

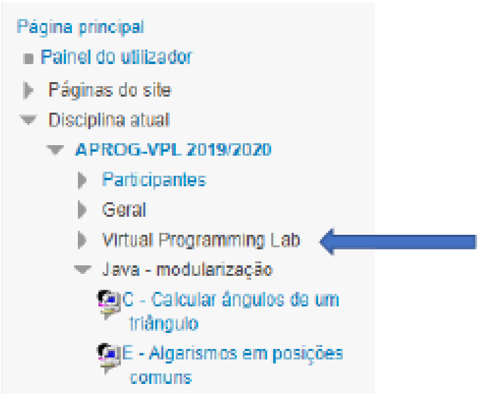
O VPL é um plugin do Moodle desenvolvido gestão de tarefas de programação, permitindo testar programas e fornecer uma avaliação imediata.

Deverão ser verificadas as mesmas restrições mencionadas para o uso do Mooshak, nomeadamente:

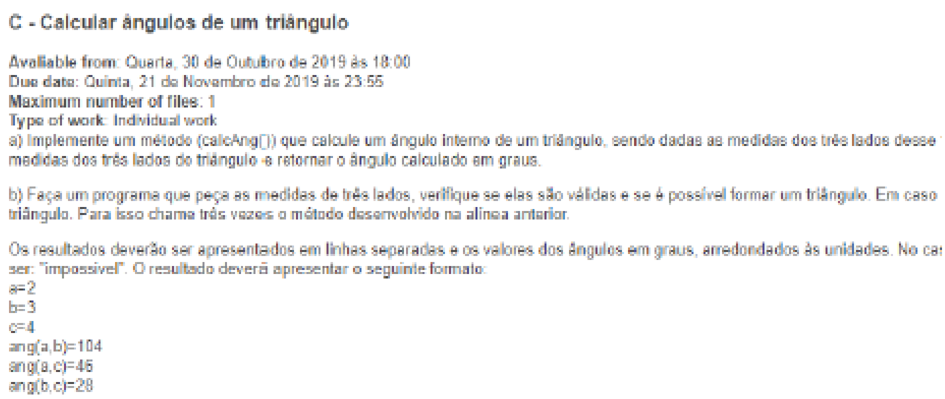
- O programa não estar associado a um package;
- Cada programa estar num único ficheiro;
- Criar apenas um objeto da classe Scanner para a leitura de dados;
- Respeitar rigorosamente o formato exigido no enunciado.

Os exercícios a testar são alguns dos que já foram testados no Mooshak.

Para proceder à submissão e teste do programa tem que aceder ao exercício em causa, por exemplo, o exercício C – Calcular ângulos de um triângulo.



Após a seleção irá diretamente para o enunciado do exercício a que corresponderia escolher a opção **Description**



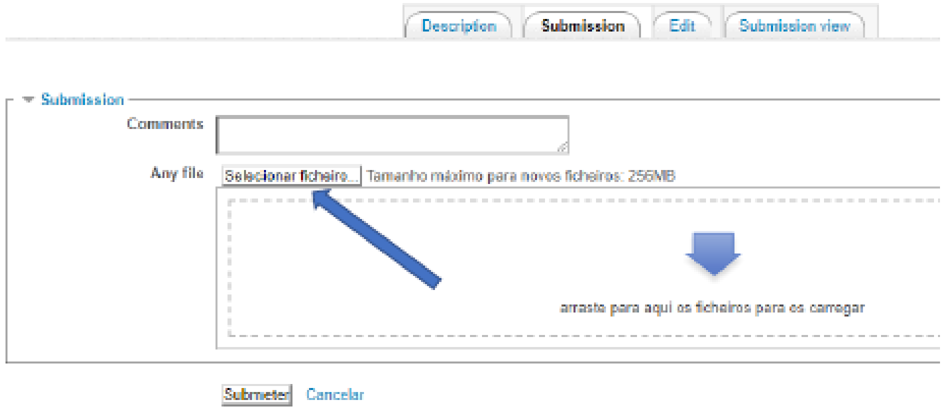
C - Calcular ângulos de um triângulo

Available from: Quarta, 30 de Outubro de 2019 às 18:00
Due date: Quinta, 21 de Novembro de 2019 às 23:55
Maximum number of files: 1
Type of work: Individual work
a) Implemente um método (calcAng()) que calcule um ângulo interno de um triângulo, sendo dadas as medidas dos três lados desse
medidas dos três lados do triângulo e retornar o ângulo calculado em graus.

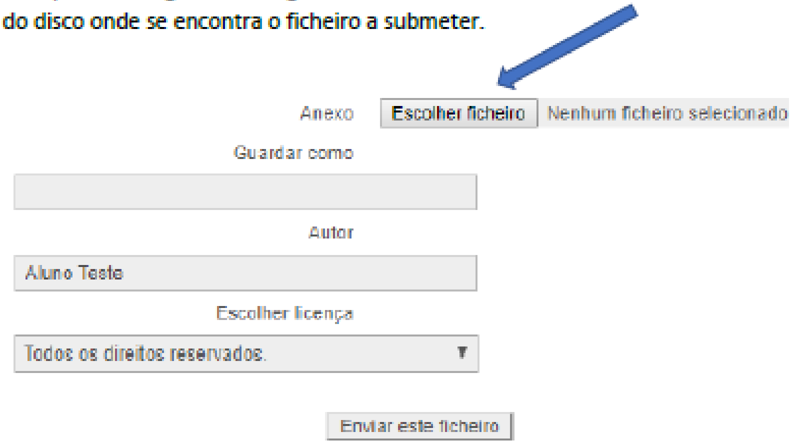
b) Faça um programa que peça as medidas de três lados, verifique se elas são válidas e se é possível formar um triângulo. Em caso
triângulo. Para isso chame três vezes o método desenvolvido na alínea anterior.

Os resultados deverão ser apresentados em linhas separadas e os valores dos ângulos em graus, arredondados às unidades. No ca
ser: "impossível". O resultado deverá apresentar o seguinte formato:
a=2
b=3
c=4
ang(a,b)=104
ang(a,c)=46
ang(b,c)=28

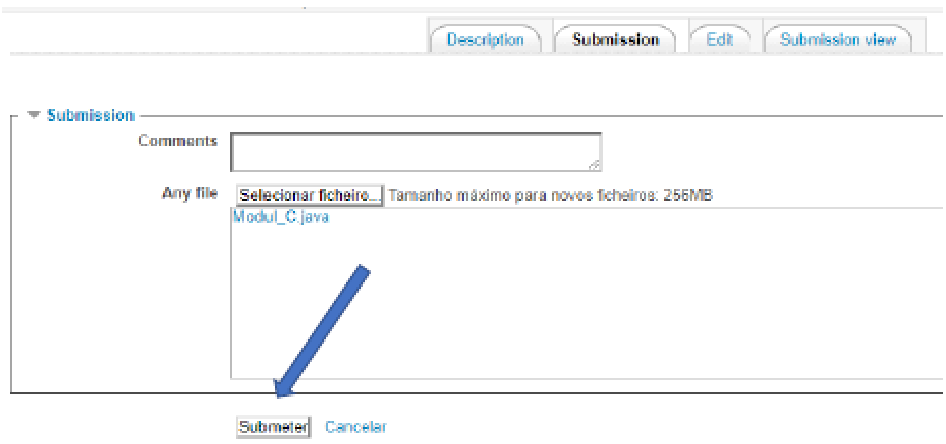
Escolhendo a opção Submission, surge o seguinte ecrã



Para proceder à submissão pode arrastar o ficheiro para a zona assinalada, escolher "Selecionar ficheiro" o que fará surgir o ecrã seguinte onde deverá selecionar "Escolher ficheiro" e indicar o local do disco onde se encontra o ficheiro a submeter.



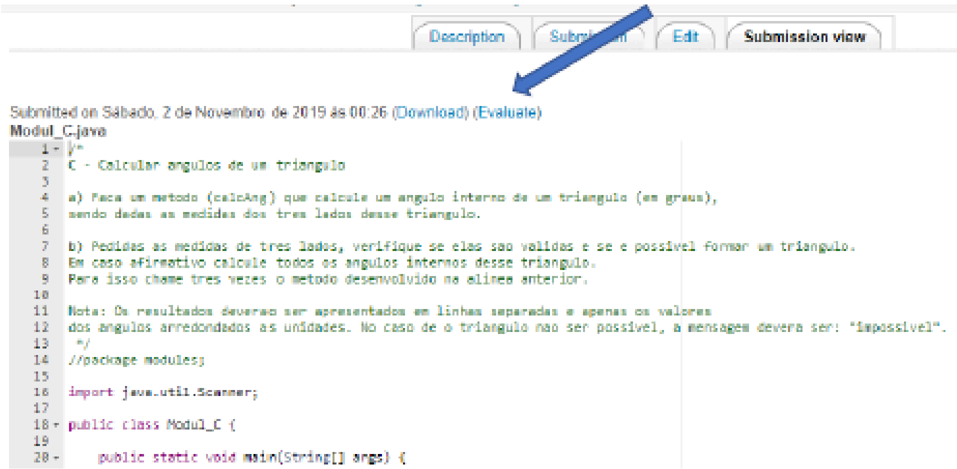
Depois de selecionado o ficheiro deve escolher "Submeter".



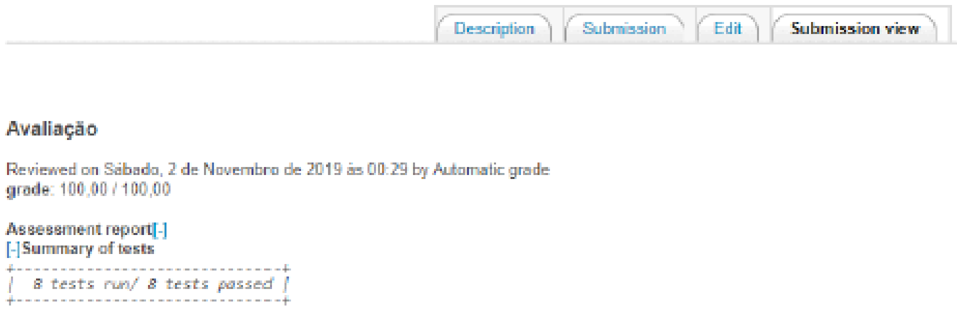
isep Instituto Superior de Engenharia do Porto

moodle Virtual Programming Lab

Após a submissão aparece o código. Para o testar pode escolher Evaluate.

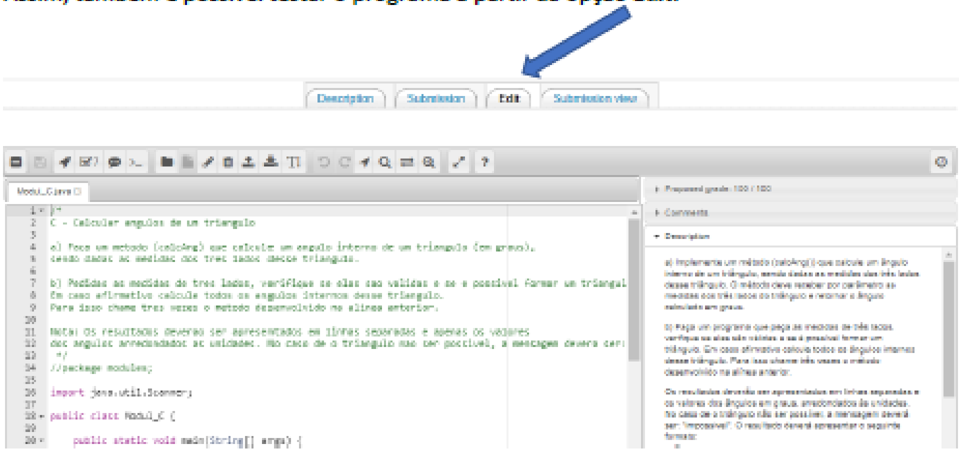


Assim o programa será testado com um conjunto de dados previamente definidos e gerado um relatório com o resultado dos testes efetuados.



O VPL tem um editor sendo possível codificar diretamente onde alterar o código submetido.

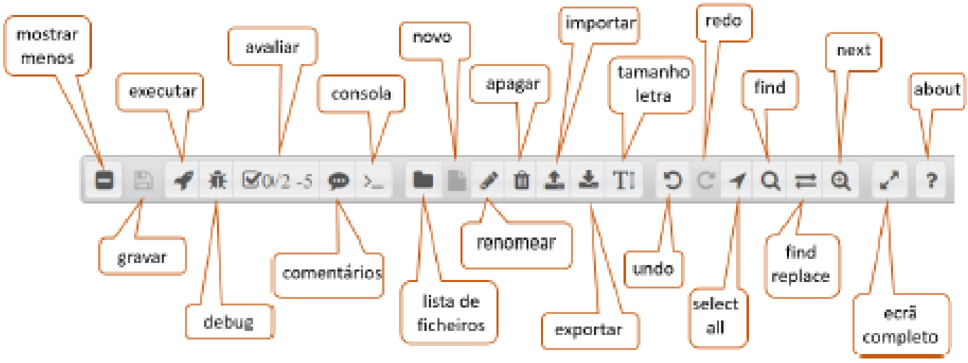
Assim, também é possível testar o programa a partir da opção Edit.



O menu de edição, execução e avaliação disponibiliza as seguintes opções (algumas das quais, na sua vista, poderão não estar visiveis):

Instituto Superior de Engenharia do Porto

moodle Virtual Programming Lab



É possível executar o programa, compilando e testando com valores introduzidos pelo utilizador, selecionando executar.

Para testar com os casos de testes deverá ser escolhida a opção avaliar, sendo apresentado o resultado no quadro à direita do código.